ΠΑΤΡΑ ΙΑΝΟΥΑΡΙΟΣ 2020

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«ΑΝΑΠΤΥΞΗ WEB ΕΦΑΡΜΟΓΩΝ ΜΕ ΠΡΟΟΔΕΥΤΙΚΗ ΜΕΘΟΔΟ» “PROGRESSIVE WEB APPS”

ΣΤΕΦΑΝΟΣ ΚΟΚΚΑΛΗΣ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ: ΓΙΑΝΝΗΣ ΤΖΗΜΑΣ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή

Πάτρα, Ημερομηνία

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Ονοματεπώνυμο, Υπογραφή
2. Ονοματεπώνυμο, Υπογραφή
3. Ονοματεπώνυμο, Υπογραφή

Υπεύθυνη Δήλωση Φοιτητή

Βεβαιώνω ότι είμαι συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επίσης βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για τη συγκεκριμένη εργασία.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Πανεπιστημίου Πελοποννήσου δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα εκ μέρους του Τμήματος.

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία τ__ φοιτητ__ _____ που την εκπόνησε. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης ο συγγραφέας/δημιουργός εκχωρεί στο Πανεπιστήμιο Πελοποννήσου, μη αποκλειστική άδεια χρήσης του δικαιώματος αναπαραγωγής, προσαρμογής, δημόσιου δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσής τους διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος και για όλο το χρόνο διάρκειας των δικαιωμάτων πνευματικής ιδιοκτησίας. Η ανοικτή πρόσβαση στο πλήρες κείμενο για μελέτη και ανάγνωση δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, αποθήκευση, πώληση, εμπορική χρήση, μετάδοση, διανομή, έκδοση, εκτέλεση, «μεταφόρτωση» (downloading), «ανάρτηση» (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού. Ο συγγραφέας/δημιουργός διατηρεί το σύνολο των ηθικών και περιουσιακών του δικαιωμάτων.

Αφιέρωση

Στη μνήμη του παππού μου,
του Στέφανου

Ευχαριστίες

Αρχικά, θέλω να ευχαριστήσω τους συμφοιτητές μου για τη συντροφικότητα στο φοιτητικό ταξίδι και για την κοινή αναζήτηση, τόσο στο αντικείμενο σπουδών, όσο και στη ζωή.

Ευχαριστώ τους μέντορες μου, συναδέλφους και ταυτόχρονα φίλους μου Τάσο Γκιούρη και Ματθαίο Φόνσο, για την καθοδήγηση που μου παρείχαν στη βαθιά κατανόηση του WEB καθώς και την γνώση που απέκτησα.

Ευχαριστώ την Κατερίνα και τους φίλους που ψύχραιμα σεβάστηκαν τις απαιτήσεις μου μέχρι την ολοκλήρωση της εργασίας.

Επίσης, θέλω να ευχαριστήσω τον καθηγητή μου κ. Γιάννη Τζήμα για την επίβλεψη της παρούσας πτυχιακής εργασίας.

Τέλος, θέλω να ευχαριστήσω την οικογένειά μου για την υπομονή και την στήριξή τους όλα αυτά τα χρόνια.

Περίληψη

Στην παρούσα πτυχιακή εργασία παρουσιάζεται μία σύγχρονη κατηγορία εφαρμογών και συγκεκριμένα εκείνη των Progressive Web Applications, καθώς προτείνονται τρόποι σχεδιασμού και ανάπτυξης Web εφαρμογών (εφαρμογών δηλαδή που εκτελούνται μέσα από κάποιο πρόγραμμα περιήγησης του Παγκόσμιου Ιστού) με χαρακτηριστικά όπως αυτά που κατέχουν οι native εφαρμογές (οι εφαρμογές δηλαδή που είναι κατασκευασμένες να εγκαθίστανται και να λειτουργούν, εξ' ολοκλήρου και συγκεκριμένα, σε ένα λειτουργικό σύστημα και μια συσκευή έξω από κάποιο πρόγραμμα περιήγησης). Αρχικά, αναφέρεται η ίδρυση του παγκόσμιου ιστού καθώς και η επαναστατική μετάβαση του από τη στατική δομή στη δυναμική, προσφέροντας τεράστιες δυνατότητες και αίσθηση διαδραστικότητας στους χρήστες. Στη συνέχεια, γίνεται ανάλυση των απαιτήσεων που πρέπει να πληρεί μια εφαρμογή για να καθοριστεί ως ένα Progressive Web Application, καθώς γίνεται και επισκόπηση των απαραίτητων τεχνολογιών που χρησιμοποιούνται για την κατασκευη του. Έπειτα, αναλύονται οι διαδικασίες αποθήκευσης πόρων στη κρυφή μνήμη του προγράμματος περιήγησης με σκοπό την άμεση φόρτωση τους και την εκτέλεση της εφαρμογής σε περίπτωση εργασίας εκτός σύνδεσης. Τέλος, κατασκευάζεται μια εφαρμογή με προοδευτική μέθοδο περιέχοντας αυθεντικοποίηση χρηστών, εύχρηστη πλοήγηση, προσαρμοζόμενο σχεδιασμό και λειτουργικότητα σε πραγματικό χρόνο. Η εφαρμογή πέραν του γεγονότος ότι θα τρέχει στο πρόγραμμα πείγησης, θα μπορεί επίσης να εγκασταθεί σε κινητές συσκευές και να λειτουργήσει χωρίς σύνδεση στο διαδίκτυο εώς έναν βαθμό, χάρις την κρυφή μνήμη.

Λέξεις κλειδιά: PWA, NodeJS, Κρυφή Μνήμη, Αποκρισιμότητα, Ειδοποιήσεις Ώθησης, Εργασία εκτός σύνδεσης, Αυθεντικοποίηση, Native αισθητική

Abstract

This thesis presents a modern class of applications, the Progressive Web Applications. It proposes ways of designing and developing web applications (applications running through a browser) with features such as those of native applications (applications that are designed and developed to be installed and get operated entirely on a specific operating system and device outside of a browser). At first, the founding of the worldwide web is mentioned as well as its revolutionary transition from static to dynamic stracture, offering enormous potential and a sense of interactivity to the users. This thesis also analyzes the requirements an application must meet so it can be categorized as a Progressive Web Application, as well as it presents an overview of the modern technologies that are used to develop a modern app. Next, the resource storage procedures in the browser's cache are analyzed in order to instantly load resources and execute the application in offline sessions. Finally, a Progressive Web Application is built and it features user authentication, easy navigation, responsive design and real-time functionality.
The application works on web browsers. Also, it can be installed on mobile devices and can operate offline to some degree.

Keywords: PWA, NodeJS, Cache, Responsive, Push Notifications, Offline, Authentication, Native

Περιεχόμενα

Αφιέρωση....4
Ευχαριστίες....5
Περίληψη....6
Abstract....8
Περιεχόμενα....10
Εισαγωγή Στο WEB....13
0.1 Ο Παγκόσμιος Ιστός....13
0.2 Browsers και Servers....14
0.3 Πρωτόκολλο μεταφοράς HTTP....15
0.4 Web Services....15
0.5 Πώς λειτουργούν τα προγράμματα περιήγησης....15
0.6 Εμπειρία Χρήστη....17
ΚΕΦΑΛΑΙΟ 1....20
ΚΕΦΑΛΑΙΟ 2....25
ΚΕΦΑΛΑΙΟ 3....29
3.1 App Manifest....30
3.2 Service Worker....31
ΚΕΦΑΛΑΙΟ 4....35
4.1 ECMAScript 6+....35
4.2 Promises API....35
4.3 Fetch API....36
4.4 Messaging API....37
4.5 Push API....42
4.6 Notification API....43
ΚΕΦΑΛΑΙΟ 5....47
5.1 Cache Storage....47

5.2 Caching Μέσα Στον Service Worker 50
ΚΕΦΑΛΑΙΟ 6 60
ΚΕΦΑΛΑΙΟ 7 67
ΚΕΦΑΛΑΙΟ 8 73
8.1 App Manifest 73
8.2 Εκτελώντας την Εφαρμογή Σε Κινητές Συσκευές Από Localhost 74
8.3 Safari App Icons 78
ΚΕΦΑΛΑΙΟ 9 82
9.1 Επισκόπηση Τεχνολογιών 82
9.2 Εγκατάσταση 83
9.3 Δημιουργώντας το ‘App Shell’ 88
9.4 Προσθήκη Λειτουργικότητας 91
9.5 Κάμερα 95
9.6 Κατασκευάζοντας τα μηνύματα 101
9.7 Συγχρονίζοντας τα μηνύματα μέσω του socket.io 103
9.8 Προσθήκη Service Worker 111
9.9 Προσθέτοντας Αυθεντικοποίηση Χρήστη 116
9.9.1 Database Connection 117
9.9.2 HBS - View Engine for Handlebars.js 118
9.9.3 Register 124
9.9.4 Login (Update User Session, Return Auth Cookie) 130
9.9.5 Profile View Και Logout 135
9.10 Push Notifications 142
9.11 App Manifest Και Testing 155
Βιβλιογραφία 162
Πίνακας Συντομογραφιών 166

Εισαγωγή Στο WEB

Το Δίκτυο Που Συνεχώς Επεκτείνεται

Το 1969, το Υπουργείο Άμυνας των Ηνωμένων Πολιτειών χρηματοδότησε την δημιουργία του ARPANET, μια πρώιμη μορφή του Διαδικτύου. Το ARPANET (The Advanced Research Projects Agency Network) αντιπροσωπεύει το Δίκτυο Υπηρεσιών Προηγμένων Ερευνητικών Προγραμμάτων με σκοπό την απομακρυσμένη και γρήγορη ανταλλαγή πληροφοριών. Το ARPANET ήταν το δίκτυο από συνδεδεμένα μεταξύ τους κέντρα υπερυπολογιστών που τα διαχειρίζονταν κυβερνητικά γραφεία και πανεπιστήμια.

Οι οργανισμοί αυτοί ήθελαν να συνδέσουν τα μεμονωμένα δίκτυά τους για μεταφορά πληροφοριών μεγάλης κλίμακας. Ωστόσο, πολλοί από αυτούς ακολούθησαν διαφορετικά πρότυπα και τεχνικές εφαρμογές. Στη δεκαετία του 1970 δημιουργήθηκε το πρωτόκολλο ελέγχου μετάδοσης και το πρωτόκολλο Διαδικτύου, γνωστό ως TCP / IP, για την παροχή προτύπων-στάνταρντς γύρω από τη μεταφορά δεδομένων σε μορφή πακέτων που θα επέτρεπε σε αυτά τα πρώιμα δίκτυα να επικοινωνούν μεταξύ τους.

Το πρωτόκολλο TCP / IP ερευνήθηκε και προσδιορίστηκε σε όλη τη δεκαετία του 1970 και εγκρίθηκε στις αρχές της δεκαετίας του 1980. Καθώς τα διαφορετικά δίκτυα υιοθέτησαν το TCP / IP, δημιουργήθηκε το διασυνδεδεμένο παγκόσμιο δίκτυο δικτύων που σήμερα είναι γνωστό ως Διαδίκτυο.

0.1 Ο Παγκόσμιος Ιστός

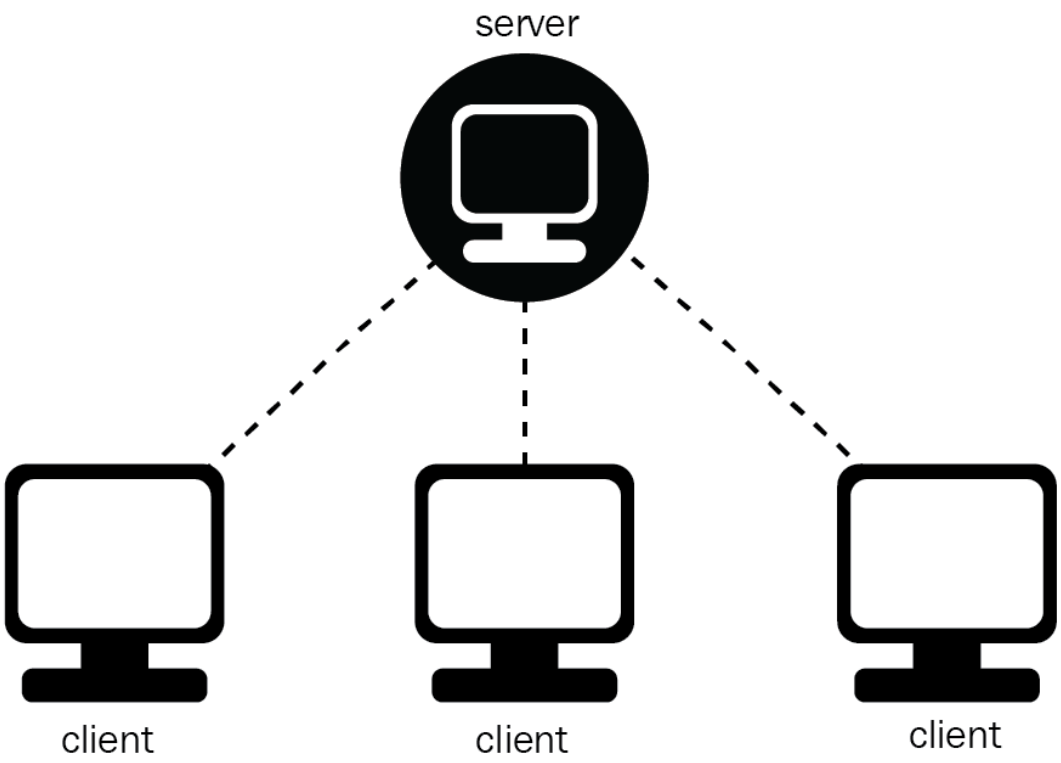
Ενώ οι άνθρωποι χρησιμοποιούν συχνά τους όρους **Διαδίκτυο (internet)** και **Παγκόσμιο Ιστό (world wide web)** ελάχιστοι κατανοούν πως αναφέρονται σε διαφορετικά θέματα.

Το **Διαδίκτυο** αναφέρεται στο *πραγματικό δίκτυο* συνδεδεμένων υπολογιστικών συσκευών. Για τον χρήστη μιας υπολογιστικής συσκευής όμως, δεν υπήρχε ένας διαισθητικός τρόπος να περιηγηθεί στο διαδίκτυο. Έτσι κι αλλιώς, το Διαδίκτυο έχει σκοπό απλά να μεταφέρει μηνύματα που παράγει ένας υπολογιστής και να τα παρουσιάζει σε έναν άλλον.

Η επαφή του ανθρώπου με το Διαδίκτυο άλλαξε το 1989 όταν ο Tim Berners-Lee επινόησε τον **Παγκόσμιο Ιστό**. Ο Παγκόσμιος Ιστός είναι μια συλλογή αλληλοσυνδεδεμένων ιστοτόπων (websites) και άλλων πόρων ιστού. Ο Παγκόσμιος Ιστός, σε συνδυασμό με την άνοδο των προγραμμάτων περιήγησης ιστού (browsers) τη δεκαετία του 90, εισήγαγε ένα φιλικό προς το χρήστη περιβάλλον που του επέτρεψε να περιηγείται στο περιεχόμενο πολυμέσων και να αλληλεπιδρά με άλλους χρήστες.

Η εφεύρεση του Παγκόσμιου Ιστού οδήγησε στη χρήση του Διαδικτύου στην ευρύτερη κοινωνία μέσα από τη δεκαετία του 1990 και τη δημιουργία μιας ποικιλίας ιστοσελίδων που εξακολουθούν να χρησιμοποιούνται μέχρι και σήμερα.

0.2 Browsers και Servers



Όπως αναφέρθηκε, το Διαδίκτυο είναι ένα δίκτυο που συνδέει συσκευές υπολογιστών παγκοσμίως επιτρέποντας στους ανθρώπους να μοιράζονται πληροφορίες μεταξύ τους παρά τις τεράστιες αποστάσεις. Αλλά πώς στέλνονται οι πληροφορίες από τη μια συσκευή στην άλλη;

Ένας τρόπος κατανόησης αυτής της διαδικασίας είναι να εξεταστεί το **μοντέλο πελάτη-διακομιστή (client - server)**. Σε αυτό το μοντέλο, ο *πελάτης (client)* αναφέρεται στη συσκευή ή το πρόγραμμα του χρήστη που **υποβάλλει αίτημα** για δεδομένα. Ένας client μπορεί να είναι ένας browser ή μια εφαρμογή που εκτελείται σε μια συσκευή χρήστη (υπολογιστή, φορητό υπολογιστή, smartphone ή tablet)

Ο *διακομιστής (server)* είναι η συσκευή ή το πρόγραμμα σε αυτό το δίκτυο που **περιμένει για τα εισερχόμενα αιτήματα** και **στέλνει πίσω τις απαντήσεις** με τα δεδομένα. Αυτό μπορεί να είναι

ένας εσωτερικός διακομιστής, ένας ενοικιαζόμενος διακομιστής σε ένα κέντρο δεδομένων ή ένας σύγχρονος cloud server.

0.3 Πρωτόκολλο μεταφοράς HTTP

Το **HTTP** σημαίνει HyperText Transfer Protocol. Είναι το πρωτόκολλο που χρησιμοποιείται από τον Παγκόσμιο Ιστό και αυτό καθορίζει τον τρόπο με τον οποίο μορφοποιούνται οι πληροφορίες, καθώς και με ποιον τρόπο μεταδίδονται και ποιες είναι οι ενέργειες ενός διακομιστή και του προγράμματος περιήγησης σε διάφορες καταστάσεις. Ο client που υποβάλλει αιτήματα, το κάνει διότι ζητάει να του επιστρέψει δεδομένα ο server ή για να στείλει ο ίδιος ο client δεδομένα στον server. Αυτές οι δύο κοινές μορφές αιτήματος αναγνωρίζονται από το HTTP ως **GET** και **POST** μέθοδοι, αντίστοιχα. Όταν ένας server αποκρίνεται σε κάποιον client, ο server ορίζει έναν κωδικό κατάστασης ως μέρος της απόκρισης.

Οι κωδικοί κατάστασης υποδεικνύουν εάν ή όχι το αίτημα HTTP ολοκληρώθηκε με επιτυχία. Εάν παρουσιάστηκε κάποιο σφάλμα περιέχουν πληροφορίες σχετικά με το είδος του σφάλματος που συνέβη. Ο κωδικός κατάστασης βοηθά το πρόγραμμα περιήγησης να γνωρίζει πώς να χειρίζεται τα δεδομένα απόκρισης. Για παράδειγμα, ο κωδικός κατάστασης **404 NOT FOUND**.

Το **HTTPS** είναι ουσιαστικά το HTTP με προσθήκη λειτουργιών ασφαλείας. Δημιουργήθηκε για τις ανάγκες μεταφοράς ευαίσθητων δεδομένων στο Διαδίκτυο και την επίλυση σεναρίων υποκλοπής τους. Είναι το προεπιλεγμένο πρωτόκολλο για τη διεξαγωγή οικονομικών συναλλαγών και μπορεί να προστατεύσει τους χρήστες ενός ιστότοπου από τη λογοκρισία μιας κυβέρνησης ή από έναν ISP. Προκειμένου να ενεργοποιηθεί το HTTPS χρησιμοποιείται ένα πιστοποιητικό σχετικά με το επίπεδο ασφαλούς μεταφοράς (Transport Layer Security ~ TLS) / Secure Sockets Layer (SSL).

0.4 Web Services

Μια υπηρεσία ιστού είναι μια μονάδα διαχειριζόμενου κώδικα που μπορεί να ενεργοποιηθεί εξ' αποστάσεως χρησιμοποιώντας HTTP αιτήματα. Οι υπηρεσίες Web επιτρέπουν την προβολή των λειτουργιών του υπάρχοντος κώδικα μέσω του δικτύου. Αφού εκτεθεί στο δίκτυο, μία άλλη εφαρμογή μπορεί να χρησιμοποιήσει τη λειτουργικότητα του προγράμματος.

0.5 Πώς λειτουργούν τα προγράμματα περιήγησης

Μέχρι στιγμής καλύφθηκε πως **ένα αίτημα** και **μία απάντηση** χρησιμοποιούνται μεταξύ ενός **client** και ενός **server**. Αλλά τις περισσότερες φορές, οι συσκευές δεν κάνουν ένα μόνο αίτημα.

Κάθε φορά που φορτώνεται μια ιστοσελίδα, η συσκευή στέλνει ένα αίτημα στον server για κάθε αρχείο/πόρο που συνθέτει τη συγκεκριμένη σελίδα. Έτσι, ακόμη και όταν φορτώνεται μόνο μια ιστοσελίδα, αυτή η σελίδα μπορεί να κάνει πολλαπλά αιτήματα για να ανακτήσει διάφορα κομμάτια περιεχομένου, όπως για παράδειγμα τις εικόνες.

- *Πώς λειτουργεί αυτή η διαδικασία όταν γίνονται ταυτόχρονα πολλαπλά αιτήματα;*

Όλα τα παρακάτω βήματα συμβαίνουν σε λιγότερο από ένα δευτερόλεπτο:

Όταν ένας χρήστης πληκτρολογεί μια διεύθυνση URL στο πρόγραμμα περιήγησης και πατάει το κουμπί ‘Enter’, ο server επεξεργάζεται το αίτημα και στέλνει το αρχείο HTML πίσω στον client. Τα αρχεία HTML συνθέτουν το περιεχόμενο ενός ιστοτόπου και περιέχουν συνδέσμους για πρόσθετα στοιχεία που απαιτούνται για την προβολή του ιστοτόπου. Το πρόγραμμα περιήγησης θα αρχίσει να αναζητεί στοιχεία στο αρχείο HTML και θα κάνει πρόσθετα αιτήματα HTTP για οποιουσδήποτε άλλους εξωτερικούς πόρους που χρησιμοποιούνται από το αρχείο HTML. Αυτό περιλαμβάνει συχνά:

• Ένα ή περισσότερα αρχεία CSS (Cascading Style Sheets). Η CSS δημιουργεί την εμφάνιση και τη διάταξη μιας ιστοσελίδας. Το πρόγραμμα περιήγησης θα ζητήσει το αρχείο CSS και όταν το στέλνει ο διακομιστής, το πρόγραμμα περιήγησης αναλύει το αρχείο αυτό και αρχίζει να εφαρμόζει τα οπτικά στυλ στο περιεχόμενο του ιστότοπου.

• Ο κύκλος αιτήματος-απόκρισης αποστέλλει επίσης στοιχεία ιστού, όπως εικόνες σε μορφή jpeg,png,gif και βίντεο, από τον server στο πρόγραμμα περιήγησης. Αν αυτά τα αρχεία είναι μεγάλα μπορεί να υπάρξει μια αξιοσημείωτη καθυστέρηση προτού να αναπαραχθούν από το πρόγραμμα περιήγησης.

• Ένα ή περισσότερα αρχεία JavaScript. Η JavaScript κάνει τη σελίδα διαδραστική. Αυτή η γλώσσα προγραμματισμού λειτουργεί ως η "συμπεριφορά" της ιστοσελίδας. Μια ιστοσελίδα που δεν χρησιμοποιεί JavaScript είναι γνωστή ως στατική ιστοσελίδα.

Στα περισσότερα σύγχρονα προγράμματα περιήγησης, αυτά τα πρόσθετα αιτήματα γίνονται παράλληλα. Αυτό σημαίνει ότι τα αιτήματα αυτά ξεκινούν την ίδια στιγμή και το πρόγραμμα περιήγησης δεν περιμένει να λάβει έναν πόρο πριν ζητήσει τον επόμενο. Όλοι οι πόροι εμφανίζονται το συντομότερο δυνατό. Η δομή της σελίδας μέσα από το HTML αρχείο θα εμφανίζεται ακόμα και αν κάποια από τα υπόλοιπα στοιχεία δεν έχουν ληφθεί.

Φυσικά, γίνεται στοχοποίηση μερικών διαδικασιών εώς να φορτωθεί τελικά ο ιστότοπος. Αγνοήθηκαν αρκετές σύνθετες διαδικασίες, όπως για παράδειγμα η υπηρεσία αποκωδικοποίησης του URL (από string μορφή σε public IP) από τον DNS server που παρέχει ο ISP. Το γεγονός αυτό συνέβη διότι γίνεται εστίαση μόνο στην διαδραστικότητα μεταξύ client-server, αποφεύγοντας την επιπρόσθετη πολυπλοκότητα.

Ο χρήστης μπορεί πλέον να αλληλεπιδρά με τον ιστότοπο που ζήτησε να εισέλθει. Η ταχύτητα μέχρι την ολοκλήρωση της παραπάνω διαδικασίας είναι ανάλογη με την ταχύτητα της σύνδεσης του χρήστη, με το μέγεθος του ιστοτόπου, με τη φυσική απόσταση μεταξύ του browser και του server, με το πόσους clients εξυπηρετεί ο server (traffic) τη στιγμή εκεινή καθώς και με την ισχύ του.

0.6 Εμπειρία Χρήστη

Όλο και περισσότερες επιχειρήσεις που παρέχουν web υπηρεσίες ή εκείνες που διαφημίζουν το προϊόν τους μέσα από το web, με τον καιρό καταλαβαίνουν πως για να παραμείνουν ανταγωνιστικές είναι απαραίτητο να επενδύσουν στην εμπειρία των χρηστών.

Η εμπειρία χρήστη (User Experience/UX) αναφέρεται στα συναισθήματα και τη στάση ενός ατόμου σχετικά με τη χρήση ενός συγκεκριμένου προϊόντος, συστήματος ή υπηρεσίας.

Η εμπειρία των χρηστών είναι ένας φιλοσοφικός όρος και πρόκειται να κατευθύνει το μέλλον του web που γνωρίζουμε σήμερα. Μπορεί να θεωρηθεί υποκειμενική, πρόκειται δηλαδή εώς έναν βαθμό για την ατομική αντίληψη και σκέψη σε σχέση με ένα σύστημα, όμως δεν είναι. Η αντικειμενικότητα έγκειται στο κοινό σύστημα αξιών που αφορά τη βελτιστοποίηση της ποιότητας της επικοινωνίας μεταξύ ανθρώπου-υπολογιστή. Η εμπειρία του χρήστη είναι δυναμική, καθώς μεταβάλλεται συνεχώς με την πάροδο του χρόνου λόγω των μεταβαλλόμενων συνθηκών χρήσης.

Περιλαμβάνει τις πρακτικές, βιωματικές, συναισθηματικές, σημαντικές και πολύτιμες πτυχές της αλληλεπίδρασης ανθρώπου-υπολογιστή και της ιδιοκτησίας του προϊόντος. Επιπλέον, περιλαμβάνει τις αντιλήψεις του ατόμου σχετικά με τις πτυχές του συστήματος όπως για παράδειγμα τη χρησιμότητα, την ευχρηστία και την αποδοτικότητα.

<u>Απαιτήσεις που πρέπει να συναντήσει ένα προϊόν για μέγιστο UX</u>:

**** Μπορεί να χρησιμοποιηθεί; ****

Μία από τις πιο βασικές απαιτήσεις του UX είναι στην πραγματικότητα να βρίσκεται σε θέση το προϊόν να πετύχει αυτό που του έχει τεθεί να κάνει. Εάν ένα λογισμικό έχει σφάλματα στον προγραμματισμό του (bugs) ή στην περίπτωση που δεν μπορεί να χρησιμοποιηθεί γιατί δεν υποστηρίζεται από μία συσκευή, έχει δηλαδή πρόβλημα συμβατότητας (compatibility issues) τότε το προϊόν είναι άχρηστο.

** **Μπορεί να ανακαλυφθεί;** **

Η εύρεση των πληροφοριών που χρειάζεται ένας χρήστης είναι επίσης σημαντική. Είναι το μενού πλοήγησης σε ένα λογισμικό διαισθητικό; Είναι η γραμμή αναζήτησης στη συνηθισμένη εώς σήμερα θέση και εκεί που πρέπει να είναι για να είναι ορατή όταν ο χρήστης αποφασίσει να τη χρησιμοποιήσει; Εάν πρέπει ο χρήστης να σκεφτεί πολύ για το πώς να βρει αυτό που ψάχνει τότε το UX αγνοείται. Το ίδιο πρέπει να ισχύει και στο να βρεθεί το ίδιο το λογισμικό, είτε αναζητώντας στο web είτε μέσα από κάποιο app store.

** **Εξυπηρετεί κάποια ανάγκη του χρήστη;** **

Ένα προϊόν μπορεί να είναι όμορφα σχεδιασμένο και εύκολο στη χρήση αλλά εάν δεν βοηθά στην αντιμετώπιση μιας ανάγκης που έχει κάποιος χρήστης δεν πρόκειται να τον ενδιαφέρει.

** **Εμπνέει ενδιαφέρον;** **

Εάν το σχέδιο ενός προϊόντος είναι διαισθητικό και ευχάριστο στη χρήση, ένας χρήστης θα θελήσει να το χρησιμοποιήσει. Ακόμη όμως, το πιο χρήσιμο και λειτουργικό προϊόν μπορεί να προσφέρει μια κακή εμπειρία χρήστη εάν δεν είναι φιλικό απέναντι του ή ο χρήστης δεν έχει κανένα κίνητρο να το χρησιμοποιήσει.

**** Θεωρείτε πολύτιμο; ****

Εάν ένα προϊόν δεν προσφέρει κάποια αξία πιθανότατα δεν θα χρησιμοποιείται για πολύ χρόνο. Το προϊόν εξοικονομεί χρόνο ή χρήμα; Βοηθάει το χρήστη να επιτύχει προσωπικούς ή επαγγελματικούς του στόχους; Όποια και αν είναι η τιμή αναφοράς του προϊόντος, είναι απαραίτητο για εκείνο να προσθέτει κάποια αξία στο χρήστη όταν εκείνος το χρησιμοποιεί.

**** Είναι αξιόπιστο; ****

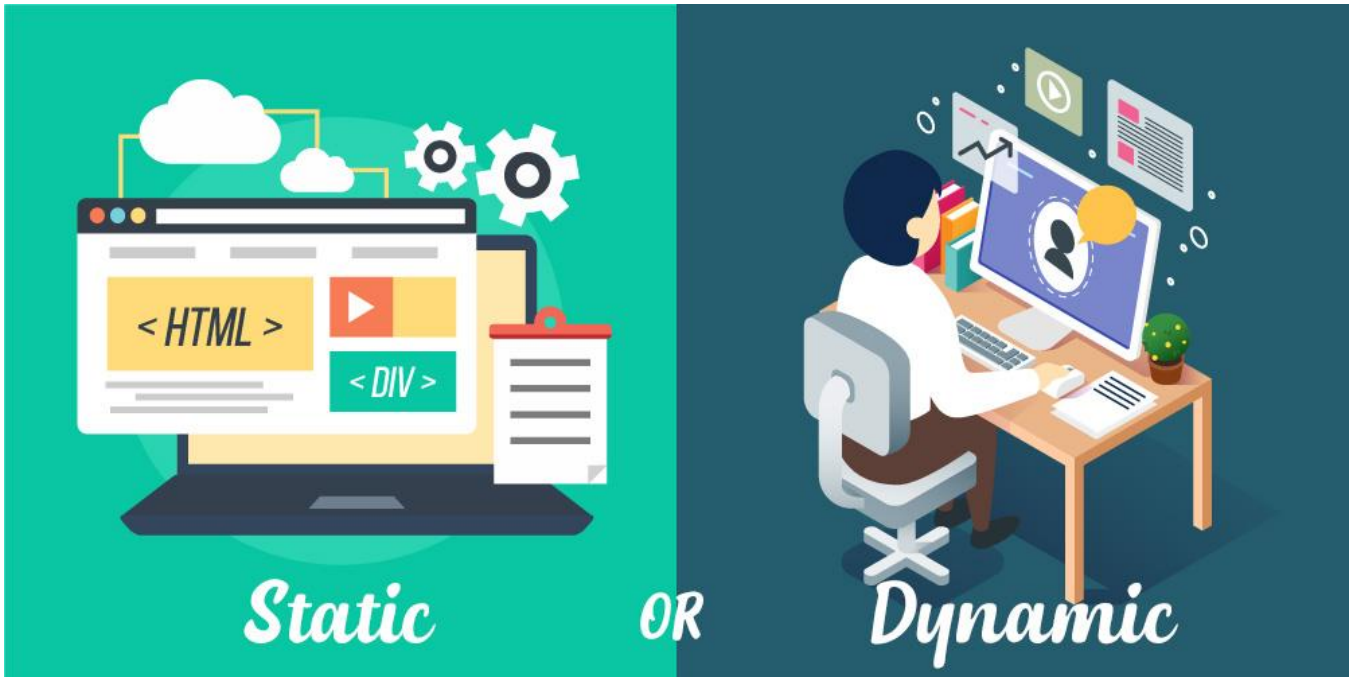
Η αξιοπιστία είναι ένα τεράστιο ζήτημα στον τεχνολογικό κλάδο. Εάν ο χρήστης δεν εμπιστεύεται έναν ιστότοπο δεν θα του προσφέρει σε καμία περίπτωση τα στοιχεία της πιστωτικής του κάρτας ώστε να πραγματοποιηθεί μια αγορά.

**** Είναι εύκολα προσβάσιμο; ****

Εάν ο χρήστης δεν μπορεί να φτάσει στον προορισμό του, δεν μπορεί να χρησιμοποιήσει τα αγαθά ή τις υπηρεσίες που προσφέρονται εκεί. Πρέπει λοιπόν, να ληφθούν υπόψη όλες οι περιπτώσεις προσβασιμότητας για να εξασφαλιστεί ότι όλοι οι χρήστες έχουν πρόσβαση στο προϊόν.

ΚΕΦΑΛΑΙΟ 1

Από Τη Στατικότητα Στη Δυναμικότητα



Όσο το διαδίκτυο επεκτείνεται κατά τη διάρκεια των χρόνων ο αριθμός των χρηστών γίνεται όλο και μεγαλύτερος. Τα τελευταία χρόνια παρατηρήθηκε μεγάλη άυξηση στους χρήστες έξυπνων κινητών συσκευών (smartphones). Σήμερα περισσότερο από το 60% της συνολικής χρήσης του διαδικτύου γίνεται μέσω κινητών συσκευών (mobile). Οι χρήστες mobile ξοδεύουν το μεγαλύτερο μέρος του χρόνου τους στις **native εφαρμογές**, εφαρμογές δηλαδή που αναπτύχθηκαν για εξ'ολοκλήρου χρήση σε συσκευές mobile, τα λεγόμενα native apps που μπορούν να έχουν πρόσβαση και να χρησιμοποιούν άμεσα το hardware της συσκευής όπου είναι εγκατεστημένα, όπως επίσης και να λειτουργούν offline εώς ένα βαθμό συγκριτικά με τα web applications (web apps), εφαρμογές δηλαδή που αναπτύχθηκαν για χρήση μέσα από τον browser σε έναν ιστότοπο χωρίς να γίνει εγκατάσταση των εφαρμογών αυτών στο λειτουργικό σύστημα. Στην πραγματικότητα, οι mobile users ξοδεύουν το 87% του χρόνου τους σε native apps έναντι του 13% των χρηστών σε καποιον mobile browser (πηγή: *The 2017 US Mobile App Report* [*α*]). Αυτό οφείλεται στο γεγονός ότι τα native apps είναι ανεπτυγμένα για να συνδέονται με την καθημερινή ζωή των χρηστών και είναι πιο προβλέψιμα. Είναι εύκολο δηλαδή, να ανακαλυφθούν και να εγκατασταθούν μέσα από μια περιήγηση στο app store, να βρεθούν στην αρχική οθόνη της κινητής συσκευής και να είναι άμεσα προσβάσιμα.

Καθώς οι ταχύτητες σύνδεσης και οι τεχνολογίες στο web προχώρησαν, όλο και πιο σύνθετες αλληλεπιδράσεις γίνονταν πιθανές.

Δεδομένου ότι γύρω από το 2005 οι **τεχνολογίες ανάπτυξης ιστού** έχουν μετατοπιστεί από την στατική στην δυναμική αρχιτεκτονική, υπάρχει μεγάλο ενδιαφέρον στο μοντέλο χρήσης του web αλλά και στον προγραμματισμό του. Με το πέρασμα των χρόνων οι ιστότοποι φαίνεται πως μετατρέπονται από ένα κοντέινερ στατικού περιεχομένου εικόνων και κειμένου, σε ιστοσελίδες που προσφέρουν διαδραστικότητα στη χρήση και που αλλάζουν βάσει συμβάντων (events) ή συνθηκών (conditions), προσφέροντας ένα νέο τεράστιο εύρος δυνατοτήτων στον τομέα του development με στόχο την μέγιστη απόδοση στην εμπειρία χρήσης. Ένα μόνο κλικ ή κάποια μετακίνηση του κέρσορα σε μια ιστοσελίδα μπορεί να προκαλέσει την παραγωγή, την απόκρυψη ή την ολική αλλαγή περιεχομένου. *Προγραμματιστικά* υπάρχουν δύο τρόποι για να δημιουργηθεί το δυναμικό αυτό αποτέλεσμα:

- Χρησιμοποιώντας *client-side scripting* μέσω της γλώσσας JavaScript και συγκεκριμένα με την βοήθεια της τεχνολογίας **AJAX** (Asynchronous JavaScript And XML) επιτρέπει την *ασύγχρονη μεταφορά δεδομένων* στο μοντέλο client-server, ως απάντηση σε συγκεκριμένα events. Σε αυτή την περίπτωση, η δυναμικότητα παρουσιάζεται άμεσα κατά την προβολή του ιστοτόπου αφού είναι εφικτό να γίνει ανταλλαγή δεδομένων μεταξύ του server και του client χωρίς ο χρήστης να ανανεώσει τη σελίδα. Να εμφανιστεί για παράδειγμα περιεχόμενο κατα την παραλαβή νέων δεδομένων απο τον server στον client ή αντίστοιχα, ή να παρουσιάζονται ανανεωμένες τιμές σε έναν πίνακα μιας σελίδας μετά από κάποιο προγραμματισμένο χρονικό περιθώριο που πραγματοποιούνται AJAX κλήσεις.

- Χρησιμοποιώντας *server-side scripting* μέσω της γλώσσας **PHP** για την εκτέλεση και την χειραγώγηση των αιτημάτων(requests) και αποκρίσεων(responses) μεταξύ ενός server και πολλών clients ώστε να επιτευχθεί η δυναμικότητα στη διαδραστικότητα του χρήστη με τον ιστότοπο. Μπορούν να προγραμματιστούν εκτελέσιμες SQL κλήσεις στη βάση δεδομένων και να ανανεωθεί, για παράδειγμα, ο προβαλλόμενος HTML κώδικας σε πραγματικό χρόνο ή να αλλάξει η παρεχόμενη προέλευση ιστοσελίδας και η προσαρμογή της ακολουθίας των σελίδων που απαρτίζουν τον ιστότοπο ή να αλλάξει κάποιο αόρατο περιεχόμενο ιστού όπως κάποιος header που παρέχεται στον browser. Τα requests και τα responses που λαμβάνει/στέλνει ένας διακομιστής(server) μπορούν να καθοριστούν από όρους, όπως τα δεδομένα σε παραμέτρους της διεύθυνσης URL ή τα δεδομένα που αφορούν το access-control και τον έλεγχο ταυτότητας ή ο τύπος του προγράμματος περιήγησης που χρησιμοποιεί ο χρήστης.

Στο πρώιμο web ενα user input συνήθως θα οδηγούσε τον χρήστη σε μια νέα σελίδα και θα έπρεπε να περιμένει μέχρι να φορτωθεί. Η **jQuery** ήταν η πρώτη βιβλιοθήκη της JavaScript που έκανε εύκολο τον προγραμματισμο του front-end ενός ιστοτόπου, όπως επίσης και των κλήσεων AJAX. Η jQuery μπορεί να ανακτήσει τα δεδομένα εύκολα ενώ η ιστοσελίδα εκτελείται.

Στη συνέχεια, γεννήθηκε η ιδέα των SPAs (*Single Page Applications)* χάρις τη βιβλιοθήκη της **AngularJS** που επιτρέπει την πλοήγηση χωρίς επαναφόρτωση από τη μία προβολή-σελίδα (page as view) στην άλλη καθώς εκτελούνται διεργασίες στο παρασκήνιο. Η Angular είναι βασισμένη στην αρχιτεκτονική MVC (model-view-controller).

Για εύκολη και γρήγορη δυναμική εμπειρία χρήσης και έχοντας περιεχόμενο που ανταποκρίνεται στο input του χρήστη χωρίς επαναφόρτωση της σελίδας χρησιμοποιούνται βιβλιοθήκες που αφορούν το front-end ενός ιστοτόπου, όπως η Bootstrap που καθιστά το κομμάτι της δόμησης και του styling σελίδων εύκολα και γρήγορα. Η **Bootstrap** περιέχει πρότυπα σχεδίασης CSS και JavaScript για interface components όπως τυπογραφία, φόρμες, κουμπιά, πλοήγηση κ.α.. Επίσης, η Bootstrap ωθεί τον developer να δώσει έμφαση στην οικοδόμηση ανταποκρινόμενων σε διαφορετικές οθόνες (**responsive**) ιστοσελίδων που αποδίδονται από ένα σύστημα πλέγματος (grid system). Ο client που επισκέπτεται έναν ιστότοπο μπορεί να χρησιμοποιεί mobile, tablet, laptop ή desktop συσκευή. Σε αυτό το εύρος μεγέθους οθόνης στην προβολή του web, το grid system έγινε απαραίτητο και πασίγνωστο στους web devs. Η φιλοσοφία πίσω από το προσφερόμενο grid βασίζεται στο γεγονός ότι μέσα από διαθέσιμες HTML κλάσεις που είναι αντιστοιχισμένες με συγκεκριμένο styling στο CSS αρχείο της Bootstrap, η προβολή της σελίδας κβαντοποιείται σε στοιχεία-δείγματα καθιστώντας το μέγεθος των δειγμάτων που προκύπτουν σε σχετικές μονάδες όπως ποσοστά (π.χ. max-width: 90%) και όχι απόλυτες μονάδες όπως pixels (π.χ. max-width: 1920px). *Ο responsive σχεδιασμός ιστοσελίδων είναι μια σύγχρονη μέθοδος κλειδί για τη μέγιστη απόδοση στο user experience*. Είναι η προσέγγιση που υποδηλώνει ότι ο σχεδιασμός και η ανάπτυξη πρέπει να ανταποκρίνονται στις ανάγκες των χρηστών και στις συσκευές που χρησιμοποιούνται από εκείνους (με βάση το μέγεθος και τον προσανατολισμό της οθόνης).

Υπήρξαν αρκετές ακόμη σημαντικές τεχνικές αναβαθμίσεις που επέτρεψαν καθεμία από αυτές τις προόδους στο web ανά τα χρόνια. Από το 2009 αναπτύχθηκε η βιβλιοθήκη **NodeJs** η οποία επιτρέπει τον back-end προγραμματισμό από την πλευρά του διακομιστή (server-side processing) σε JavaScript. Η JavaScript χρησιμοποιείται επίσης για τη δυναμική δημιουργία σελίδων μέσα από τον διακομιστή με τη δυνατότητα να αποστέλλεται πλήρως στους clients.

Η άνοδος των web βιβλιοθηκών που προσφέρουν υπηρεσίες σε βάσεις δεδομένων όπως η **Spring (java)**, η **Django (python)** και η **Ruby-on-Rails (ruby)** επέτρεψαν την αποτελεσματική δημιουργία, αποθήκευση και άμεση εμφάνιση περιεχομένου στο χρήστη που δημιουργήθηκε από τον ίδιο.

Παρά τις εφαρμογές που βασίζονται σε αυτές τις τεχνολογίες, σε συσκευές όπως το iPhone, που η άφιξη του έγινε το 2007, και στη συνέχεια η πρώτη γενιά smartphone με το open-source λειτουργικό σύστημα android, οι προσπάθειες για να δουλεύουν οι web εφαρμογες από τους mobile χρήστες με τον ίδιο τρόπο που τις δούλευαν μέχρι τότε οι χρήστες desktop απέτυχαν σε σύγκριση με τα native apps. Ένα **web app** είναι γενικευμένο για πολλαπλές πλατφόρμες και δεν έχει εγκατασταθεί τοπικά, όμως είναι διαθέσιμη στο web και είναι προσβάσιμη μέσα από τον browser.

Ένα **native app** αναπτύσσεται για μια συγκεκριμένη πλατφόρμα που είναι εγκατεστημένη σε μια υπολογιστική συσκευή. Τα native apps παρέχουν καλύτερη εμπειρία χρήστη και κάνουν ταχύτερα boot σε σύγκριση με την ανάγκη φόρτωσης στον browser. Τα packaged resources και η άμεση πρόσβαση στο hardware της συσκευής επιτρέπουν στα native apps να εκτελούνται πολύ γρήγορα και να παρέχουν περισσότερες δυνατότητες από ότι τα web apps. Μέχρι τα μέσα του 2010 ωστόσο, οι συνεχείς βελτιώσεις που εμφανίστηκαν με την άφιξη των *HTML5* και *CSS3*, οι σημαντικά πιο ικανοί και συμβατοί με τα νέα πρότυπα web browsers και σε συνδυασμό με τους ισχυρούς επεξεργαστές νέας γενιάς, κατέστησαν τις **υβριδικές εφαρμογές** (hybrid apps) μια βιώσιμη εναλλακτική λύση. Ένα hybrid app είναι εκείνο που συνδυάζει στοιχεία των native και web apps.

Πολλές πλατφόρμες έχουν προσπαθήσει να καταστήσουν δυνατή την πρόσβαση σε native δυνατότητες ενώ παράλληλα επιτρέπουν στους προγραμματιστές να δημιουργήσουν εφαρμογές με την client-side τεχνολογία του web. Με αυτόν τον τρόπο απορρίπτουν μία ή περισσότερες πτυχές του συστήματος κοινής αξίας μεταξύ προγραμματιστή και εφαρμογής, όπως για παράδειγμα την δυνατότητα τεχνικού πειραματισμού εφ όσον δεν προσφέρεται κατανόηση των web και mobile τεχνολογιών αλλά αντ' αυτού, αυτές οι πλατφόρμες προσφέρουν γρήγορες λύσης ανάπτυξης υβριδικών εφαρμογών επί πληρωμή και με τη χρήση τυποποιημένου κώδικα. Οι πλατφόρμες αυτές από τεχνικής άποψης είναι εξαιρετικές, αλλά δεν είναι οι web τεχνολογίες που επιτρέπουν την απόλυτη ελευθερία και ευελιξία επιλογών, καθώς επίσης δεν ενθαρρύνουν την κλιμάκωση δυνατοτήτων σε χρονικό βάθος (πλατφόρμες όπως: Cordova / PhoneGap και Crosswalk, BlackBerry WebWorks, WebOS, Chromium Embedded Framework, Electron). Αυτές οι πλατφόρμες, με αντάλλαγμα την native αίσθηση εφαρμογής: εργασία εκτός σύνδεσης, πρόσβαση στην αρχική οθόνη μέσω ενός launch icon, συστήματα πρόσβασης API, δυνατότητα διανομής μέσω app stores και ολική φόρτωση μετά την πρώτη εγκατάσταση, συχνά εγκαταλείπουν τη δυνατότητα διασύνδεσης.

ΚΕΦΑΛΑΙΟ 2

Εισαγωγή Στα Progressive Web Apps (PWAs)

Το 2015, ο σχεδιαστής Frances Berriman και ο μηχανικός του Google Chrome Alex Russell, εμπνεύστηκαν τον όρο '**progressive web applications**' για να περιγράψουν εφαρμογές που χρησιμοποιούν τις νέες λειτουργίες που υποστηρίζονται από το σύγχρονο πρόγραμμα περιήγησης και που καλύπτουν συγκεκριμένες προδιαγραφές που αφορούν την ευχρηστία, την απόδοση και την εμπειρία χρήστη. Αυτές οι νέες λειτουργίες είναι η **υπηρεσία εξυπηρέτησης (service worker)** και το **πρότυπο εφαρμογής (app manifest)**. Τα PWAs καθορίζονται από διάφορα χαρακτηριστικά. Σύμφωνα με τους προγραμματιστές της Google, για να καθοριστεί μία εφαρμογή ως ένα PWA, θα πρέπει να είναι:

Προοδευτική

Λειτουργεί σε οποιονδήποτε χρήστη, ανεξάρτητα από την επιλογή του προγράμματος περιήγησης και της συσκευής-λειτουργικού, επειδή είναι χτισμένη με προοδευτική μέθοδο ως βασική αρχή.

Responsive

Προσαρμόζεται σε οποιασδήποτε μορφοποίηση: mobile, tablet, laptop ή desktop οθόνη που θα παρουσιάζεται η εφαρμογή.

Ανεξάρτητη Από Την Συνδεσιμότητα

Ο service worker επιτρέπει εργασία εκτός σύνδεσης ή ακέραια εργασία σε δίκτυα χαμηλής ποιότητας.

App Like

Έχει αισθητική με αλληλεπιδράσεις και πλοήγηση σε στυλ εφαρμογής.

Up to Date

Είναι πάντα ενημερωμένη χάρις την διαδικασία ενημέρωσης των service worker.

Ασφαλής

Διανέμεται μέσω HTTPS για να αποφευχθεί το snooping και να παραμείνει εξασφαλισμένο το γεγονός ότι το περιεχόμενο δεν έχει παραβιαστεί.

Εντοπίσιμη

Μπορεί να αναγνωριστεί ως ‘εφαρμογή’ χάρη στα πρότυπα του W3C, τα app manifest αρχεία και το πεδίο εγγραφής των service workers που επιτρέπουν στις μηχανές αναζήτησης να την ανακαλύψουν.

Επανασυνδέσιμη

Επανασυνδέεται εύκολα μέσω λειτουργιών όπως οι ειδοποιήσεις ώθησης (push notifications).

Ικανότητα Εγκατάστασης

Επιτρέπεται στους χρήστες να αποθηκεύουν την εφαρμογή και να εμφανίζουν εκείνη που θεωρούν πιο χρήσιμη στην αρχική τους οθόνη χωρίς τη χρήση του app store.

Δυνατότητα σύνδεσης

Γίνεται εύκολη κοινή χρήση μέσω μιας διεύθυνσης URL.

Τα PWAs αποτελούν αναβάθμιση της υπάρχουσας web τεχνολογίας. Ως εκ τούτου, δεν απαιτούν ξεχωριστή δέσμευση ή διανομή. Η διαδικασία της δημοσίευσης ενός PWA είναι η ίδια όπως και για οποιονδήποτε άλλο ιστότοπο. Οι λειτουργίες PWA είναι συμβατές σε οποιοδήποτε πρόγραμμα περιήγησης αλλά οι λειτουργίες που έχουν μια αίσθηση native εφαρμογής, όπως η ανεξαρτησία από τη συνδεσιμότητα, η εγκατάσταση στην αρχική οθόνη και η προώθηση ειδοποιήσεων, εξαρτώνται από την υποστήριξη του προγράμματος περιήγησης. Από τον Απρίλιο του 2018, αυτές οι λειτουργίες υποστηρίζονται σε διάφορους βαθμούς από τα πιο γνωστά προγράμματα περιήγησης Microsoft Edge, Google Chrome, Mozilla Firefox και Safari.

ΚΕΦΑΛΑΙΟ 3

Επισκόπηση Τεχνολογιών Ι

Τα Progressive Web Apps γίνονται γρήγορα η λύση επιλογής των developers για native εμπειρίες. Υπάρχουν ιδέες ανάπτυξης και έννοιες που αφορούν αποκλειστικά την PWA αρχιτεκτονική. Μια παλαιότερη έρευνα καταναλωτών έδειξε ότι το ήμισυ όλων των χρηστών smartphone κατεβάζει ακριβώς 0 εφαρμογές το μήνα (πηγή: Most US Smartphone Users Download Zero Apps per Month [β]).

Ο μέσος χρήστης αγοράζει τη συσκευή smartphone, στη συνέχεια προχωράει στην εγκατάσταση εφαρμογών που χρειάζεται και χειρίζεται για την καθημερινή του χρήση. Οι smartphone χρήστες βρίσκονται συνεχώς στους web browser και τα web apps είναι μια διεύθυνση URL μακρυά. Για να χρησιμοποιήσει ένας χρήστης ένα web app πρέπει να είναι συνδεδεμένος στο διαδίκτυο. Τα web apps έχουν έλλειψη native λειτουργιών και έχουν κόστος στα δεδομένα. Τα PWA γεφυρώνουν αυτό το κενό. Προσφέρουν απλές online και offline εμπειρίες, έχουν πρόσβαση σε native δυνατότητες και είναι μαζικά προσβάσιμες. Λειτουργούν ως εφαρμογές ξεχωριστές από το πρόγραμμα περιήγησης. Ο τρόπος εργασίας των PWA επιτρέπει την προσθήκη αυτής της λειτουργικότητας σε υπάρχουσες ιστοσελίδες σε διαδοχικά/κλιμακωτά στάδια (**Progressive Enhancement**).

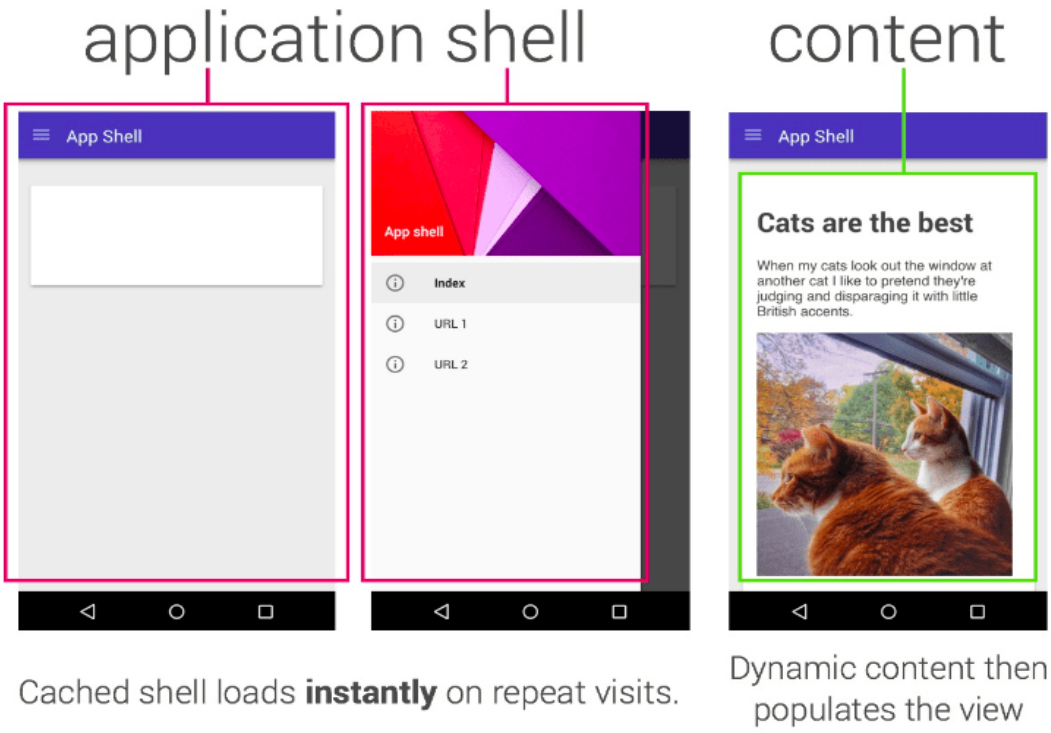
Για την μετατροπή ενός πρότυπου ιστοτόπου σε PWA, πρέπει να γίνει προσθήκη δύο επιπέδων τεχνολογίας.

Αρχικά των **service workers** - αυτόνομα JavaScript scripts που διαχειρίζονται την ιστοσελίδα. Εκτελούνται πλήρως ανεξάρτητα από τον υπόλοιπο ιστότοπο και λειτουργούν σαν αντικείμενο μεσολάβησης (Proxy Object) μέσα από τον browser. Ένας βασικός ρόλος του service worker είναι η αποθήκευση του περιεχομένου σε μια προσωρινή μνήμη και γι 'αυτό χρησιμοποιεί το νέο ‘cache API’. Εάν ο browser δεν υποστηρίζει καμία από αυτές τις λειτουργίες, ο ιστότοπος θα συνεχίσει να λειτουργεί κανονικά (**Feature Detection**).

Για να θεωρηθεί μια ιστοσελίδα PWA πρέπει να σχεδιαστεί σαν εφαρμογή (App-Like) ως πρώτη προτεραιότητα. Πρέπει να είναι responsive ή mobile first ώστε ο σχεδιασμός της να ταιριάζει με εκείνον μιας native εφαρμογής. Πρέπει επίσης, να είναι γρήγορη και να λειτουργεί ομαλά σε οποιαδήποτε συσκευή. Στον τομέα της ανάπτυξης εφαρμογών είναι πολύ σημαντικό ζήτημα η κατανάλωση δεδομένων, οπότε ένα PWA θα πρέπει να διατρέχει στο διαδίκτυο μόνο όταν το χρειάζεται.

Ένα PWA είναι απαραίτητο να εργάζεται εκτός σύνδεσης έως κάποιο βαθμό. Το concept είναι να αφαιρείται όσο το δυνατό περισσότερο περιεχόμενο μακριά από το περιβάλλον εργασίας της εφαρμογής ώστε όλα αυτά τα στατικά στοιχεία που απαρτίζουν το **app shell** (το στατικό κομμάτι της εφαρμογής με τα λιγότερα πιθανά html/css/js αρχεία που λειτουργεί σαν κέλυφος και

αγκαλιάζει το δυναμικό content) να μπορούν να παραμείνουν προσωρινά αποθηκευμένα και τα δυναμικά στοιχεία να φορτώνονται από το δίκτυο.



Η τελική προσθήκη σε μια ιστοσελίδα για να μετατραπεί σε PWA είναι το αρχείο **App Manifest**. Αυτό είναι ένα αρχείο που απλά καθορίζει το εικονίδιο της εφαρμογής εκκίνησης(**launch icon**), την οθόνη εκκίνησης (**splash screen**) και μετατρέπει μια απλή ιστοσελίδα στη δική της οντότητα που μπορεί να εγκατασταθεί στη mobile συσκευή του χρήστη και να είναι προσβάσιμη από την αρχική οθόνη.

Υπάρχουν αρκετά στάδια για την οικοδόμηση ενός PWA αλλά το σπουδαίο είναι πως όλα όσα προαναφέρθηκαν μπορούν να γίνουν προσθήκη σε ένα web app δίχως να αναιρεθεί η web φύση του. Όταν αναπτυχθεί ένα PWA δεν θα δημιουργηθεί μόνο μια μεγάλη native εμπειρία. Το πρότυπο site από όπου ξεκίνησε το app θα παραμείνει στην αρχική του κατάσταση και θα συνεχίζει να λειτουργεί μέσα από τον browser.

Πιο αναλυτικά, οι τεχνολογίες που χρησιμοποιούνται για τη δημιουργία ενός PWA είναι οι εξής:

3.1 App Manifest

Το app manifest είναι μια προδιαγραφή του W3C που καθορίζει ένα πρότυπο βασισμένο σε JSON διάταξη ώστε να παρέχει στους προγραμματιστές μια κεντρική θέση με σκοπό να ορίσουν εκείνοι τα μεταδεδομένα που σχετίζονται με ένα web app. Τα μεταδεδομένα συμπεριλαμβάνουν:

- Το όνομα της εφαρμογής
- Σύνδεση με τα εικονίδια της εφαρμογής web ή αντικείμενα εικόνας
- Η προτιμώμενη διεύθυνση URL για την εκκίνηση ή το άνοιγμα της εφαρμογής
- Τα δεδομένα διαμόρφωσης της εφαρμογής για ορισμένα χαρακτηριστικά
- Δήλωση για τον προεπιλεγμένο προσανατολισμό της οθόνης
- Η επιλογή ρύθμισης της εμφάνισης κατά τηλειτουργίας, π.χ. 'ΠΛΗΡΗΣ ΟΘΟΝΗ'
- Splash Screen

Αυτά τα μεταδεδομένα είναι κρίσιμης σημασίας για την μετατροπή ενός web app σε PWA (προσθήκη εφαρμογής στην αρχική οθόνη, launch icon κλπ).

3.2 Service Worker

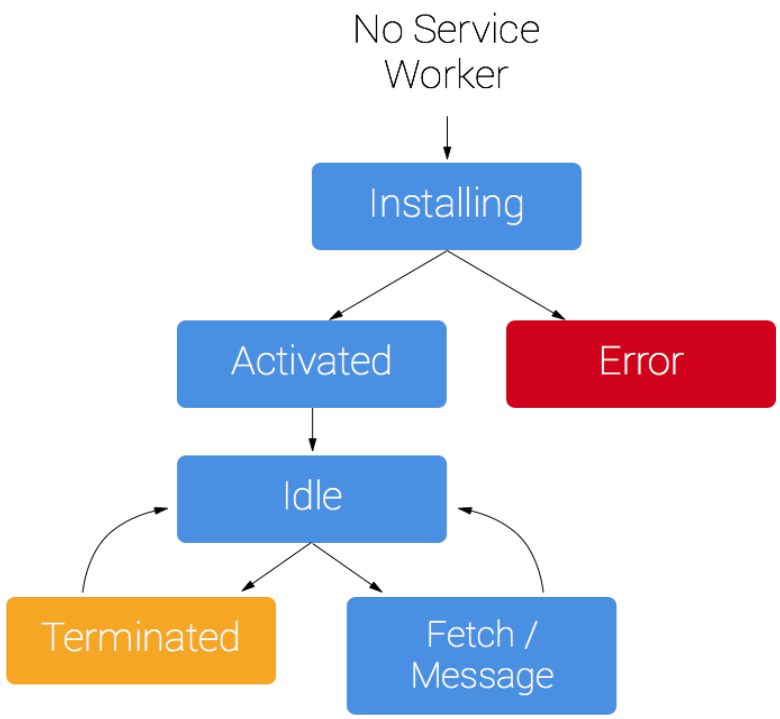
Παρέχει έναν προγραμματιστικό μεσολαβητή δικτύου στον browser για τη διαχείριση των HTTP αιτημάτων και είναι εκείνος που ευθύνεται για τη δυνατότητα offline εργασιών. Ο service worker βρίσκεται μεταξύ του δικτύου και της συσκευής για την παροχή του περιεχομένου. Είναι σε θέση να χρησιμοποιεί αποτελεσματικά τους μηχανισμούς κρυφής μνήμης (cache) που παρέχει ο browser και να επιτρέπει συμπεριφορές χωρίς σφάλματα κατά τη διάρκεια offline περιόδων. Ο service worker έχει έναν κύκλο ζωής (**life cycle**) ο οποίος είναι τελείως ξεχωριστός από την ιστοσελίδα. Πιο αναλυτικά:

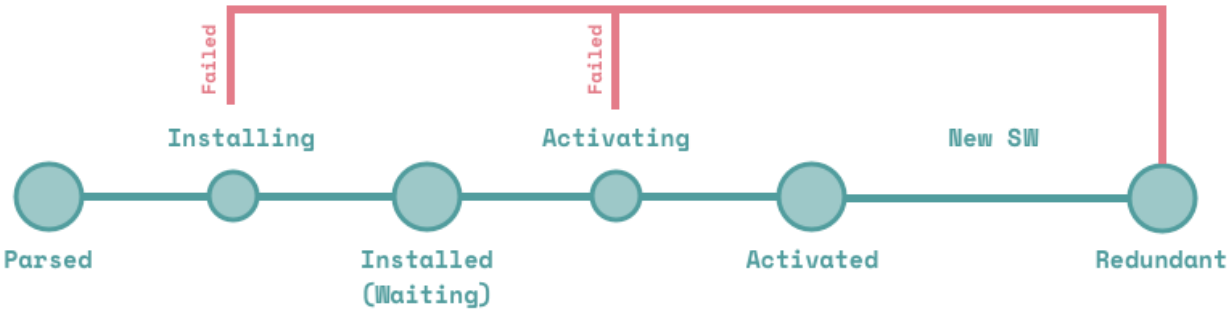
Για να **εγκατασταθεί** ένας service worker σε έναν ιστότοπο, πρέπει να καταχωρηθεί μέσα από το JavaScript αρχείο του.

Η **εγγραφή** ενός service worker στον browser θα προκαλέσει τον browser να ξεκινήσει το βήμα εγκατάστασης του στο παρασκήνιο.

Όταν εγκατασταθεί, θα ακολουθήσει το **βήμα ενεργοποίησης.** Αυτή είναι η ευκαιρία για το χειρισμό παλαιών δεδομένων στην cache.

Μετά το βήμα ενεργοποίησης, ο service worker θα έχει τον έλεγχο σε όλες τις σελίδες του ιστοτόπου που εμπίπτουν στο πεδίο εφαρμογής του, αν και η σελίδα που καταχώρησε για πρώτη φορά τον service worker δεν θα είναι υπό τον έλεγχο του μέχρι να γίνει η επαναφόρτωση της. Μόλις έχει τον έλεγχο, θα βρίσκεται σε μία από τις δύο καταστάσεις: είτε θα τερματιστεί για να εξοικονομήσει μνήμη, είτε θα χειριστεί τα fetch και message events που συμβαίνουν όταν γίνει μια αίτηση δικτύου ή σταλεί ένα μήνυμα.





Καταστάσεις του service worker:

- `Installing`
- `Installed`
- `Activating`
- `Activated`
- `Redundant`

Ιδιότητες του service worker:

- Ενεργοποιείται και διατηρείται ζωντανός αναλόγως των events
- Γενικός στη φύση του
- Προσδιορισμένη, με χρονικά περιορισμένα περιβάλλοντα, δέσμη ενεργειών
- Φυσικά endpoints για ένα ευρύ φάσμα υπηρεσιών χρόνου εκτέλεσης
- Κατανεμημένο ID ή UUID
- Ικανότητα παράλειψης των flags
- Ξεκινάει την λειτουργία του στην αρχή της εφαρμογής
- Τερματίζεται οταν δεν χρησιμοποιείται και κάνει επανεκκίνηση όταν χρειαστεί
- Κάνει εκτεταμένη χρήση των promises

Παροχές του service worker:

- Ικανότητα εύχρηστου χειρισμού των ειδοποιήσεων ώθησης (Push Notifications)
- Συγχρονισμός δεδομένων στο παρασκήνιο
- Δυνατότητα ανταπόκρισης σε αιτήματα πόρων που προέρχονται από αλλού
- Λήψη συγκεντρωτικών ενημερώσεων

ΚΕΦΑΛΑΙΟ 4

Επισκόπηση Τεχνολογιών II

Μια άλλη σημαντική πτυχή του service worker είναι το γεγονός ότι χρησιμοποιεί τα promises αντί των παραδοσιακών callbacks για την πρόβλεψη και αντιμετώπιση ασύγχρονων συμβάντων, καθώς επίσης προτιμάει την μέθοδο fetch αντί των κλήσεων AJAX και του απαρχαιωμένου XHR (XML HTTP Request). Εάν ο browser υποστηρίζει και φορτώνει τον service worker, τότε θα υποστηρίζει τα promises και το fetch. Με αυτόν τον τρόπο μπορούν να χρησιμοποιηθούν πιο νέα χαρακτηριστικά της JavaScript κατά τον προγραμματισμό μέσα στον service worker.

4.1 ECMAScript 6+

Το ECMAScript **(European Computer Manufacturers Association Script)** είναι το αρχέτυπο που καθορίζει τα standards για τις scripting γλώσσες προγραμματισμού. Η JavaScript είναι βασισμένη στο ECMAScript. Η χρήση των προσφερόμενων ιδιοτήτων που προσφέρει η ECMAScript από την έκδοση 6 (2015) μέχρι σήμερα, είναι ένας πιο σύγχρονος τρόπος scripting που επικεντρώνεται στην απόδοση κατά την ανάπτυξη web εφαρμογών. Όλα τα παρακάτω APIs, είναι πιστοποιημένα από το ECMAScript. Οι ιδιότητες που χρησιμοποιούνται συχνά είναι:

- Τα αναγνωριστικά keywords `const` και `let` αντί του αφηρημένου `var` που επιτρέπουν τη δήλωση σταθερών τιμών, καθώς και τη δήλωση μεταβλητών μέσα σε blocked scopes.
- Τα **arrow functions** που είναι ένας πιο γρήγορος τρόπος σύνταξης μεθόδων.
- Τις **κλάσεις** (Classes) που συνδυάζονται με έναν constructor και που εκείνος με τη σειρά του καλείται όποτε ένα στιγμιότυπο/object κλάσης αρχικοποιείται.

4.2 Promises API

Τα promises παρουσιάστηκαν ως ένας τρόπος καλύτερης αντιμετώπισης των κλασσικών δομών επανάκλησης (callbacks) στον ασύγχρονο κώδικα της JavaScript. Ένα promise είναι ουσιαστικά μια JavaScript class που υποδηλώνει ότι ένα object θα δώσει την έκβαση κάποιου ασύγχρονου γεγονότος, σε κάποια αόριστη χρονική στιγμή στο μέλλον.

Η βασική διαφορά με τα callbacks είναι ότι τα callbacks καλούνται όταν το ασύγχρονο γεγονός τελειώσει, ενώ τα promises απλά επιλύονται (resolve) από το ασύγχρονο γεγονός. Τα promises δηλαδή είναι βοηθοί που θα στέκονται και θα επιβλέπουν τον ασύγχρονο κώδικα και που θα εκτελεστούν όταν είναι έτοιμο ένα γεγονός να έρθει στο παρόν, στο τώρα.

Μόλις το promise δημιουργηθεί μπορεί να έχει 3 πιθανές καταστάσεις:

- `Pending`, η αρχική κατάσταση
- `Resolved`, που σημαίνει ότι η “υπόσχεση πραγματοποιήθηκε”
- `Rejected`, που σημαίνει ότι η ενέργεια απέτυχε και δεν μπορεί να πραγματοποιηθεί η υπόσχεση

Στην περίπτωση που ένα promise θα γίνει resolved μπορεί να χρησιμοποιηθεί το **.then()** στο τέλος της εκτέλεσης του, το οποίο επιστρέφει ένα άλλο promise και με αυτόν τον τρόπο μπορεί να δημιουργηθεί αλυσιδωτή αντίδραση. Στην περίπτωση που το promise γίνει rejected χρησιμοποιείται το **.catch()** για να ανιχθευθεί το error και να προγραμματιστεί η έκβαση του. Η τελευταία διαδικασία λέγεται **error handling**. Μία άλλη μέθοδος που αξίζει να αναφερθεί είναι η **.all()**, η οποία παίρνει σαν παράμετρο ένα array από promises και επιστρέφει ένα μόνο promise. Πριν εκτελεστεί θα περιμένει *όλα τα promises* να γίνουν resolved, ή κάποιο να γίνει rejected. Παρομοίως, η μέθοδος **.race()** παίρνει σαν παράμετρο ένα array από promises, όμως θα εκτελεστεί όταν διευθετηθεί *μόνο το πρώτο χρονικά promise του array*.

4.3 Fetch API

Το νέο Fetch API είναι μια συναρπαστική προσθήκη της JavaScript στο πρόγραμμα περιήγησης που αφορά την ανάκτηση πόρων και που αντικαθιστά πλήρως τα παραδοσιακά XML HTTP αιτήματα και τις κλήσεις AJAX. Το fetch παρέχει έναν γενικό ορισμό των request και response objects και άλλων που σχετίζονται με αιτήματα δικτύου. Αυτό θα τους επιτρέψει να χρησιμοποιηθούν όπου και αν χρειάζονται στο μέλλον, είτε πρόκειται για τον service worker, είτε για το API της cache, είτε για άλλα παρόμοια APIs που χειρίζονται ή τροποποιούν αιτήματα και απαντήσεις, είτε για οποιαδήποτε περίπτωση χρήσης που μπορεί να απαιτεί να δημιουργηθούν αυτοσχέδια προγραμματιστικά responses.

Παρέχει επίσης, ορισμό για σχετικές έννοιες όπως της CORS (Cross-Origin Resource Sharing) η οποία αφορά έναν μηχανισμό του web που χρησιμοποιεί κεφαλίδες προέλευσης HTTP (headers), ανιχνέυει την προέλευση των πόρων κατά τα HTTP requests / responses, επιτρέπει την επικοινωνία μεταξύ τομέων δικτύου (cross-domain communication) από τον browser, προστατεύει τον ιστότοπο από την έκθεση του σε εξωτερικά δίκτυα και αποτρέπει επιθέσεις υποκλοπής δεδομένων.

Η μέθοδος fetch() παίρνει μία υποχρεωτική παράμετρο, τη διαδρομή προς τον πόρο που θέλουμε να ανακτήσουμε. Επιστρέφει ένα promise που επιλύεται στο response του request, είτε είναι επιτυχής είτε όχι. Προαιρετικά, δέχεται ένα αντικείμενο επιλογών (options object) ως τη δεύτερη παράμετρο.

Μόλις ληφθεί μια απόκριση, υπάρχουν διάφορες διαθέσιμες μέθοδοι για να καθοριστεί ποιο είναι το περιεχόμενο και πώς πρέπει να αντιμετωπιστεί.

Ένα response object έχει την body ιδιότητα η οποία περιέχει τα δεδομένα και είναι της μορφής readable stream. Αυτή η μορφή επιτρέπει την μετατροπή της εύκολα σε διάφορους άλλους τύπους δεδομένων. Οι διαθέσιμοι τύποι είναι:

- `ArrayBuffer`
- `ArrayBufferView (Uint8Array etc)`
- `Blob/File`
- `string`
- `URLSearchParams`
- `FormData`

Μπορεί να δημιουργηθεί ένα αίτημα και μια απόκριση χρησιμοποιώντας τους constructors Request() και Response(), αλλά είναι απίθανο να γίνει αυτό άμεσα. Αντίθετα, αυτές είναι πιο πιθανό να δημιουργηθούν ως αποτελέσματα άλλων ενεργειών, για παράδειγμα χρησιμοποιώντας την FetchEvent.respondWith() μέθοδο από τον service worker.

4.4 Messaging API

Το Messaging API επιτρέπει την άμεση επικοινωνία μεταξύ δύο διαφορετικών scripts που εκτελούνται σε διαφορετικά περιβάλλοντα περιήγησης (tab ή window), επισυναπτόμενα στο ίδιο document, περνώντας μηνύματα μεταξύ τους χρησιμοποιώντας ασφαλή cross-origin κανάλια δύο οδών (σωλήνων/pipes) με μία θύρα στην κάθε άκρη. Εάν ο service worker έχει εγκατασταθεί μπορεί μέσα από το main.js για παράδειγμα, το βασικό JavaScript αρχείο της εφαρμογής που αφορά τη συμπεριφορά του client, να σταλεί message στον service worker μέσω της μεθόδου postMessage(), με τον εξής τρόπο:

```
if(navigator.serviceWorker) {
  navigator.serviceWorker.register('/sw.js').then(function(registration){
        registration.active.postMessage('Hello from
main.js');
   }).catch(console.log('SW not supported'));
}
```

Για να λάβει το μήνυμα ο service worker, **στο αρχείο του sw.js** γίνεται προσθήκη ένας eventListener με πρώτη παράμετρο το message και δεύτερη ένα callback με το ίδιο το event ως τη δική του παράμετρο. Αυτό επιτυγχάνεται ως εξής:

```
self.addEventListener('message', (e) => {
```

```
    console.log(e.data);
});
```

Η πιο συνηθισμένη περίπτωση χρήσης του Messaging API σε συνδυασμό με τα PWAs, είναι η προτροπή του user να κάνει update την εφαρμογή. Για παράδειγμα μέσα από το main.js ενημερώνεται ο χρήστης με ένα παράθυρο πως βρέθηκε αναβάθμιση της εφαρμογής. Εάν ο χρήστης συμφωνήσει με το παράθυρο και κατ'επέκταση με την εγκατάσταση του νέου service workers, θα σταλεί μήνυμα με όνομα 'update_self'. Μέσα από τον eventListener του message και το αντικείμενο 'e' που αφορά το message event, ελέγχεται το όνομα του message μέσα από την data ιδιότητα που έχει το 'e' αντικείμενο:

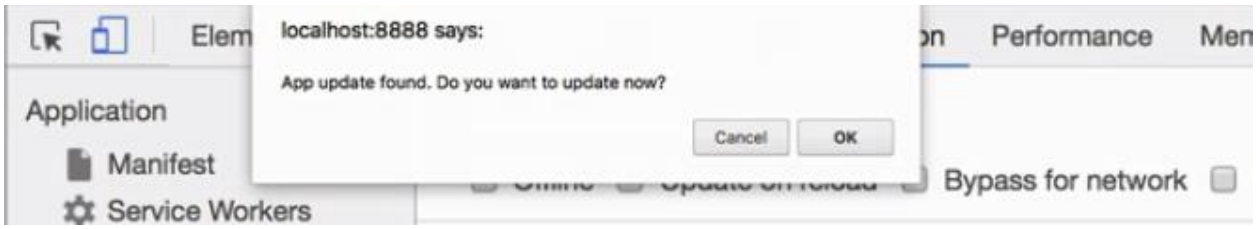
```
registration.onupdatefound = () => {
    let newSW = registration.installing;
    //Prompt User for Update
    if (confirm('App update found. Do you want to update now?')) {
        newSW.postMessage('update_self');
    }
};
```

Έπειτα για να λάβει ο νέος service worker το μήνυμα, προστίθεται στο sw.js αρχείο:

```
self.addEventListener('message', (e) => {
    if(e.data === 'update_self'){
        //Force new sw takeover from the currently active one
        self.skipWaiting();
```

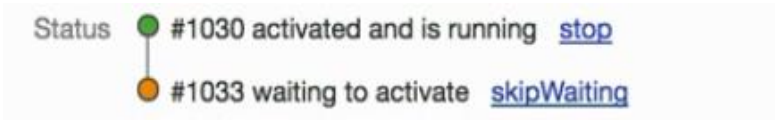
```
        console.log('Service Worker Updating');
    }
});
```

Όταν λοιπόν γίνει refresh στην εφαρμογή θα εμφανιστεί ένα pop-up παράθυρο.

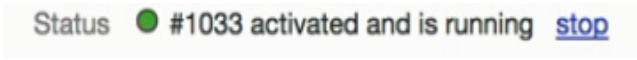


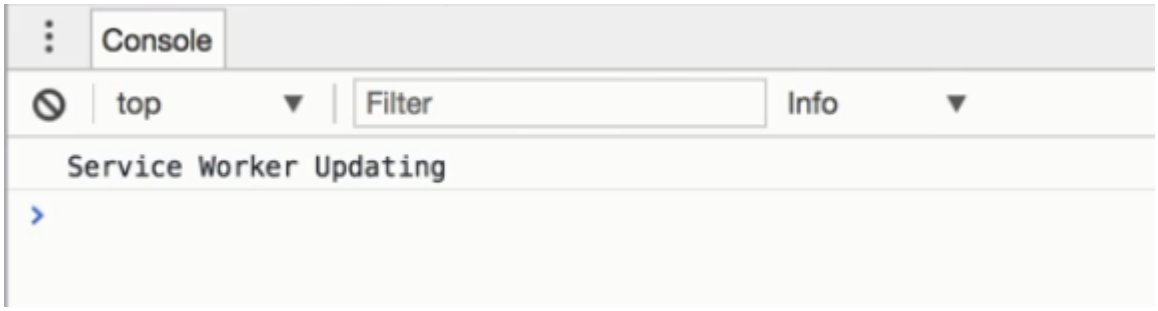
Πατώντας OK, ο νέος service worker λαμβάνει το message και μεταφέρεται από την waiting κατάσταση στην active καθώς γνωστοποιείται επίσης και από την κονσόλα. Για να εμφανιστεί η κονσόλα του browser χρείαζεται να γίνει εισαγωγή στα εργαλεία προγραμματισμού που παρέχει ο κάθε browser. Παρακάτω χρησιμοποιούνται τα devtools του Chrome.

Μετάβαση κατάστασης από:



Σε:





Αφού αναφέρθηκε ο τρόπος που ο service worker λαμβάνει messages από τον client, αναλύεται στη συνέχεια η αντίστοιχη διαδικασία για να λαμβάνει message ο client από τον service worker.

Εμβαθύνοντας στο concept του client, γίνεται ξεκάθαρο πως ένας client δεν είναι ο user. Ούτε πως ο user μπορεί να χειριστεί έναν μόνο client με χρήση μιας μοναδικής συσκευής. Όταν ο user ανοίξει την εφαρμογή μέσα από τον Chrome browser για παράδειγμα, ο Chrome εγκαθιστά τον service worker. Όταν ο ίδιος user έπειτα ανοίξει την εφαρμογή από άλλο tab στον Chrome χρησιμοποιείται ο ίδιος service worker. Σε αυτήν την περίπτωση, αναγνωρίζονται δύο client

objects από τον ίδιο service worker. Ωστόσο, εάν ο user ανοίξει την εφαρμογή σε άλλο browser (π.χ. Firefox), ο Firefox browser θα εγκαταστήσει τον service worker ανεξάρτητα για τον εαυτό του.

Ο service worker μπορεί να ανταποκρίνεται σε όλους του τους clients μέσα από το αρχείο sw.js με την εντολη **clients.matchAll()**, στοχεύοντας με αυτόν τον τρόπο όλους τους active clients. Αυτή η εντολή επιστρέφει ένα promise, που αν γίνει resolved θα υπάρξει πρόσβαση στο αποτέλεσμα μέσω της εντολής then(). Μέσα από εκείνη, θα υπάρξει πάλι πρόσβαση σε έναν πίνακα που επιστρέφεται και αποτελείται από τα objects των active clients του worker. Για την ανταπόκριση σε κάθε client ξεχωριστά, γίνεται προσπέλαση στα στοιχεία του πίνακα μέσα από την clients.forEach(client) μέθοδο, μετατρέποντας κάθε φορά που γίνεται προσπέλαση στο clients array το εκάστοτε στοιχείο clients[i], σε ένα client στοιχείο που περιέχει το αντίστοιχο object και εκείνο με τη σειρά του θα λάβει την ανταπόκριση του service worker σε μορφή message. Αυτό γίνεται με την εντολή client.postMessage(*msg*), όπου *msg* το μήνυμα σε string τύπο. Στο sw.js:

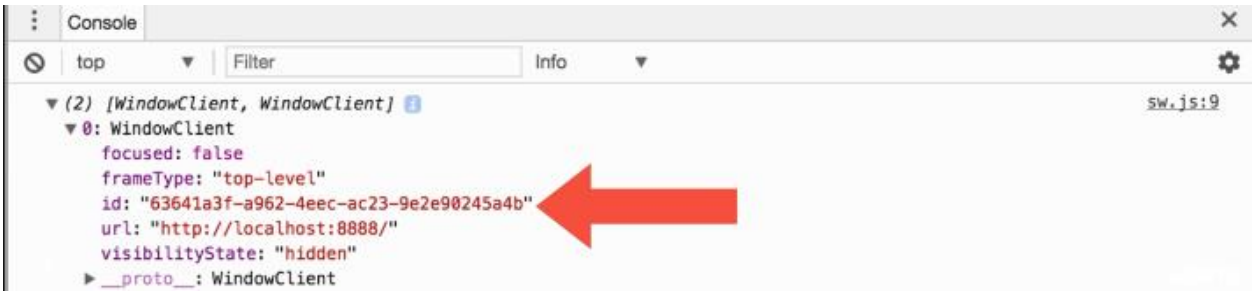
```
// Respond to all clients
self.clients.matchAll().then((clients) => {

   console.log(clients);
   clients.forEach(client) => {
      client.postMessage('Hello from Service Worker');
   });
});
```

Έπειτα πρέπει να γίνει λήψη του εισερχόμενου message μέσα από το αρχείο main.js.

```
navigator.serviceWorker.addEventListener('message', (e) => {
   console.log(e.data);
});
```

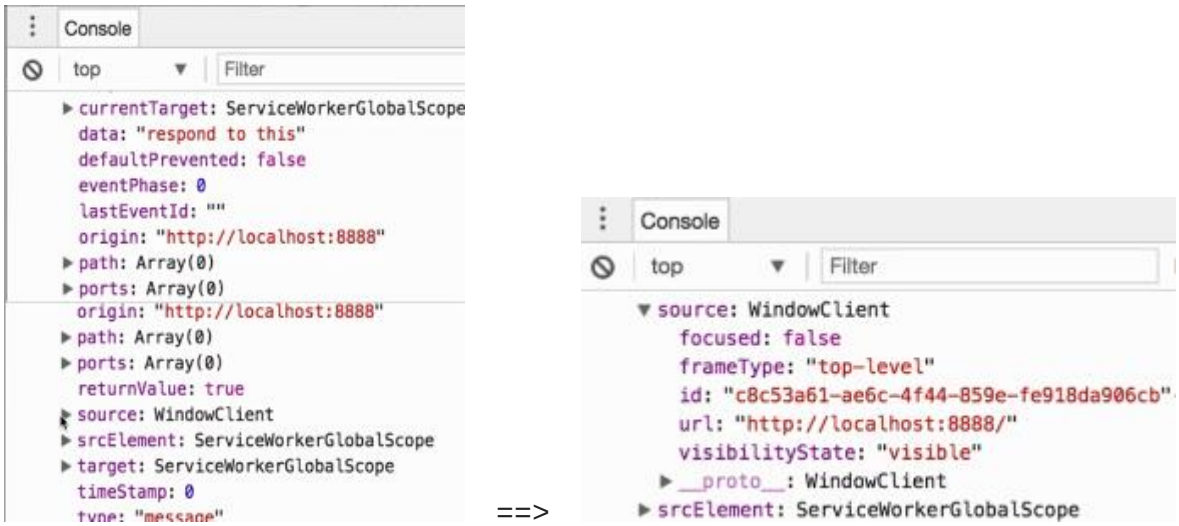
Στη συνέχεια μέσα από τον browser, κατά το update του service worker παρατηρείται στην κονσόλα το response message προερχόμενο από το main.js λειτουργώντας σαν client. Παράλληλα ανταποκρίνεται σε όλους τους clients. Ανοίγοντας ένα δεύτερο tab και τρέχοντας την εφαρμογή, και αφού ο Chrome έχει ήδη εγκατεστημένο τον service worker που χρησιμοποιείται από την ίδια σελίδα που έγινε εισαγωγή, μόλις φορτωθεί στο νέο tab, γίνεται αμέσως ένας active client του worker. Το response message στην κονσόλα θα είναι διπλό, υποδηλώνοντας πως υπάρχουν δύο διαφορετικοί active clients στον ίδιο service worker.

Στην κονσόλα θα εμφανιστούν επίσης και οι active clients και παρατηρείται ο κάθε ένας ξεχωριστά. Αυτή τη φορά η εντολή ήταν η console.log και όχι η postMessage(), για αυτόν το λόγο στην κονσόλα η επιστροφή της μεταβλητής είναι από το sw.js κατευθείαν. Μέσα στα objects του πίνακα clients παρατηρείται η id ιδιότητα αλλά με δύο ανοιχτά tabs όμως το id είναι το ίδιο.



Ο τρόπος για να επικοινωνήσει ο service worker με έναν και μόνο client από τους δύο που εξυπηρετεί, ενώ έχουν το ίδιο id, πρέπει να παρατηρηθεί το ίδιο το message event και να βρεθεί η source ιδιότητα του.

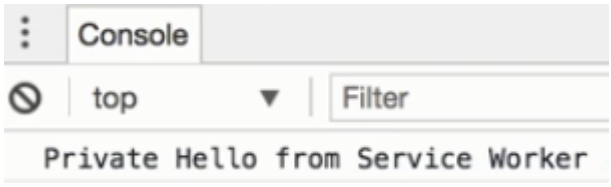
```
self.addEventListener('message, (e) => {

   //check event

     console.log(e);

});
```



Μέσα από το id του windowClient ταυτοποιείται το source του εκάστοτε client. Έτσι, για να απαντήσει ο service worker σε αυτόν τον client και μόνο, θα γίνει εισαγωγή στον κώδικα του sw.js, μέσα στην forEach λούπα, η εξής λογική:

```
//Only respond to sending client
if (e.source.id === client.id) {
   client.postMessage('Private Hello from Service Worker');
}
```

Παρατηρείται στην κονσόλα το επιτυχές message, εστάλη μία φορά και όπως προηγουμένως παρατηρείται η δυναμική προέλευση του μηνύματος από το main.js.



4.5 Push API

Το Push API παρέχει στα PWAs τη δυνατότητα λήψης μηνυμάτων που προωθούνται από έναν server, ανεξάρτητα από το αν η εφαρμογή έχει φορτωθεί τελείως σε έναν client. Αυτό επιτρέπει την παράδοση ασύγχρονων ειδοποιήσεων και ενημερώσεων σε users, γεγονός που έχει ως αποτέλεσμα την καλύτερη προσέγγιση στην εφαρμογή με νέο και έγκαιρο περιεχόμενο.

Για μια εφαρμογή που θέλει να λαμβάνει push μηνύματα απαιτείται να έχει έναν service worker εγκατεστημένο. Τότε και μόνο μπορεί η εφαρμογή να εγγραφεί για να προωθήσει ειδοποιήσεις χρησιμοποιώντας το PushManager.subscribe().

Το PushSubscription object που προκύπτει περιλαμβάνει όλες τις πληροφορίες που χρειάζεται η εφαρμογή για την αποστολή ενός push message. Δηλαδή, ένα endpoint και τα κλειδιά κρυπτογράφησης που απαιτούνται για την αποστολή δεδομένων.

Ο service worker θα ξεκινήσει και θα είναι ικανός να χειριστεί εισερχόμενα μηνύματα push, τα οποία παραδίδονται στο χειριστή συμβάντων (event handler) ServiceWorkerGlobalScope.onpush. Αυτό επιτρέπει στα PWAs να αντιδρούν στην προώθηση των μηνυμάτων που λαμβάνουν, για παράδειγμα, προβάλλοντας μια ειδοποίηση (χρησιμοποιώντας το ServiceWorkerRegistration.showNotification ().)

Κάθε συνδρομή είναι μοναδική για έναν service worker. Το τελικό σημείο της συνδρομής είναι μια μοναδική διεύθυνση URL δυνατοτήτων, καθώς η γνωστοποίηση του endpoint είναι η μόνη απαραίτητη πληροφορία για την αποστολή ενός μηνύματος στην εφαρμογή. Επομένως, η διεύθυνση URL πρέπει να παραμείνει μυστική, αλλιώς άλλες εφαρμογές ενδέχεται να είναι σε θέση να στέλνουν μηνύματα.

Η ενεργοποίηση ενός service worker για την παράδοση ενός push μηνύματος μπορεί να έχει ως αποτέλεσμα την αύξηση της χρήσης των πόρων, ιδιαίτερα της μπαταρίας μιας mobile συσκευής.

Το κάθε πρόγραμμα περιήγησης έχει διαφορετικά standards για το χειρισμό αυτού του ζητήματος. Δεν υπάρχει επί του παρόντος κανένας τυποποιημένος μηχανισμός. Ο Firefox επιτρέπει την αποστολή ενός περιορισμένου αριθμού Push messages σε μια εφαρμογή, αν και τα μηνύματα Push που παράγουν ειδοποιήσεις εξαιρούνται από αυτό το όριο. Το όριο ανανεώνεται κάθε φορά που ο user επισκέπτεται τον ιστότοπο. Αντίθετα, ο Chrome δεν έχει κανένα όριο, όμως, κάθε μήνυμα ώθησης προκαλεί αναγκαστικά την εμφάνιση μιας ειδοποίησης.

4.6 Notification API

Σε όλες τις native εφαρμογές, η επικοινωνία της εφαρμογής με τον χρήστη γίνεται μέσω μικρών παραθύρων στο interface της εφαρμογής που περιέχουν κουμπί/ά για αλληλεπίδραση με τον χρήστη. Αυτά τα παράθυρα λέγονται **notifications/ειδοποιήσεις**. Η εμφάνιση των native ειδοποιήσεων αποτελεί βασικό μέρος των περισσότερων PWAs αφού θέλουν να μιμούνται τα notifications των native apps. Μία ειδοποίηση λαμβάνεται από τα push events για να εγκαθίσταται συνήθως μια νέα ενημέρωση προερχόμενη από τον service worker. Για να εμφανιστεί η ειδοποίηση σε μια σελίδα, θα χρησιμοποιηθεί το HTML5 Notification API.

Για παράδειγμα, ως αποτέλεσμα ενός push event, μπορεί να γίνει έλεγχος εάν οι ειδοποιήσεις υποστηρίζονται από τον browser του χρήστη. Επίσης, οι ειδοποιήσεις απαιτούν την άδεια του χρήστη για να εμφανιστούν.

Οι τρεις περιπτώσεις άδειας χρήστη στον browser για την εμφάνιση ειδοποιήσεων είναι:

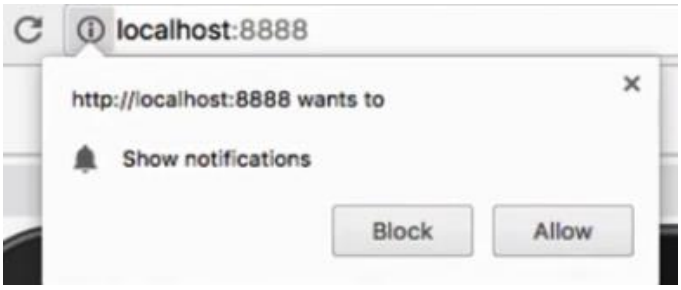
- `Granted`
- `Denied`
- `Default`

Ένας εύκολος τρόπος χρήσης του API στον κώδικα θα είναι στο main.js να προστεθεί:

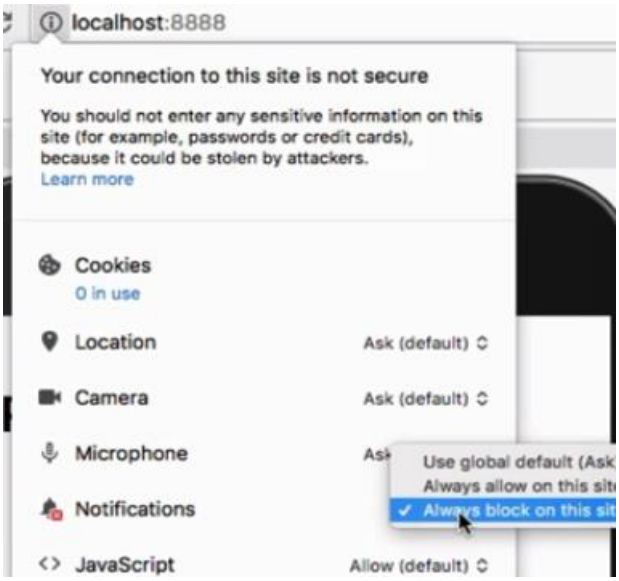
```
//Notification Support
if (window.Notification) {
    function showNotification() {
        console.log('A new Notification');
    }
    //Manage Permission
    if (Notification.permission === 'granted') {
        showNotification();
    } else if (Notification.permission !== 'denied') {
        Notification.requestPermission( (permission) => {
```

```
            if(permission === 'granted'){
                showNotification();
            }
        });
    }
}
```

Όταν τρέξει το app και γίνει reload, στον chrome θα εμφανιστεί ένα παράθυρο, απαιτώντας την άδεια του χρήστη για την εμφάνιση των ειδοποιήσεων.



Παρατηρούνται δύο επιλογές καθώς είναι γνωστό πως οι περιπτώσεις είναι τρεις.



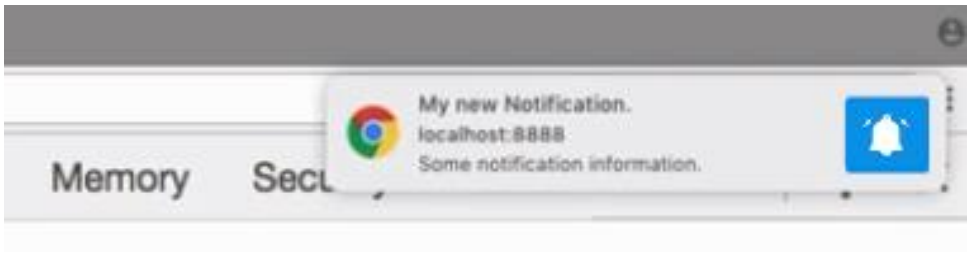
Για να ρυθμιστεί η default άδεια, πατώντας το **i** αριστερά του URL εμφανιζεται το αναδυόμενο παράθυρο με τις διαθέσιμες επιλογές και έτσι παρατηρούνται τα notifications. Κάνοντας κλικ στην επιλογή, γίνεται πρόσβαση στη ρύθμιση του default permission.

Στην περίπτωση που γίνουν 'μπλοκ οι ειδοποιήσεις' ως default επιλογή, ο user δεν θα αντιλαμβάνεται τίποτα κάθε φορά που ειδοποιήσεις θα έρχονται στον client.

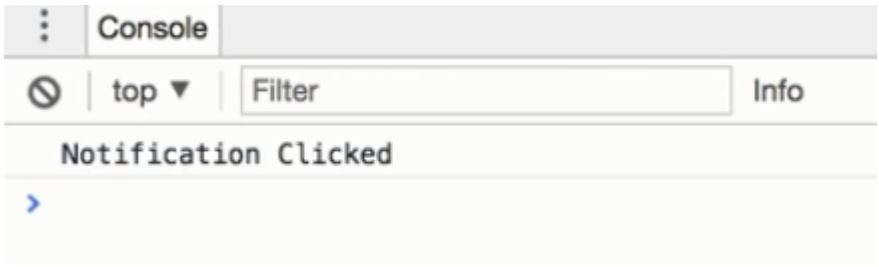
Η εμφάνιση του notification όπως και τα user interaction events πάνω στο notification, μπορούν να χειραγωγηθούν μέσα στην showNotification() λειτουργία και γίνεται ως εξής:

```
let notificationOpts = {
    body: 'Some notification information.',
    icon: '/icon.png'
}
let n = new Notification('My new Notification.', notificationOpts);
n.onclick = () => {
```

```
    console.log('Notification Clicked');
}
```



Όταν το notification πατηθεί μέσω ενός κλικ του χρήστη, θα εμφανιστεί στην κονσόλα:

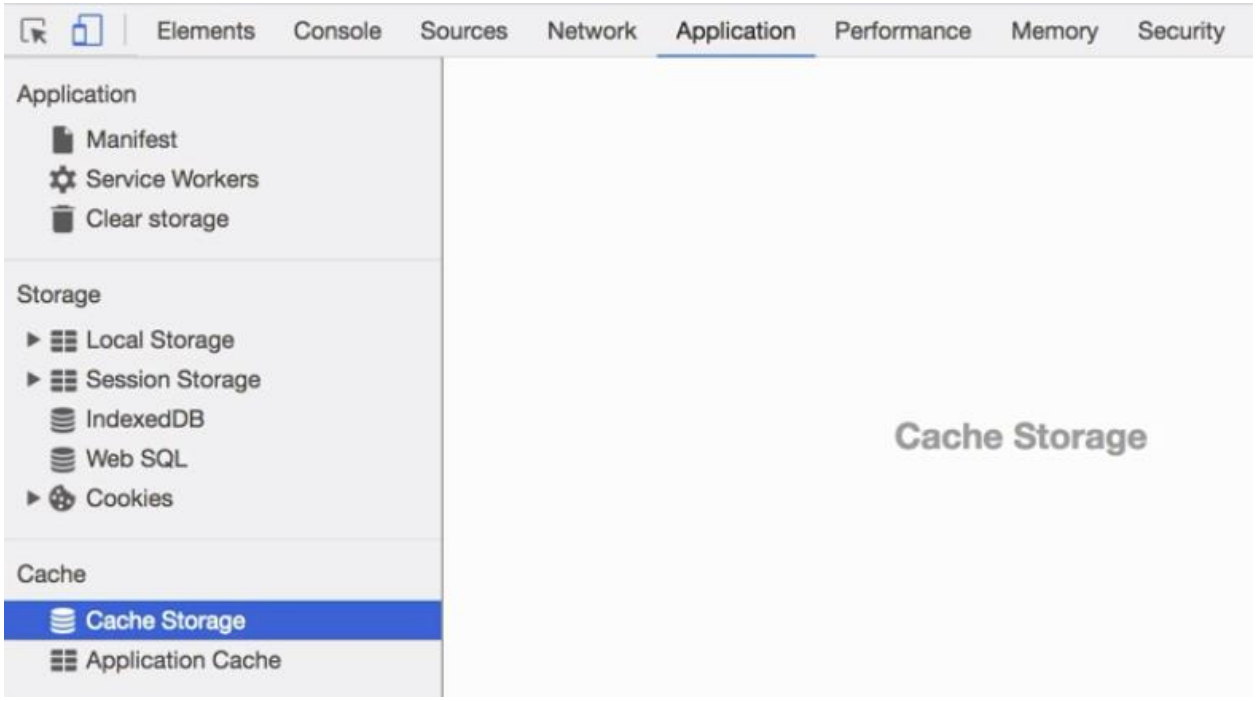


ΚΕΦΑΛΑΙΟ 5

Εμβαθύνοντας Στη Κρυφή Μνήμη

Στο κεφάλαιο αυτό αναλυεται ο κρυφός αποθηκευτικός χώρος του browser που χρησιμοποιείται από τον service worker, όχι μόνο για να λειτουργεί offline ένα PWA, αλλά και για την άμεση φόρτωση των πόρων του καθώς και την μειωμένη κατανάλωση των δεδομένων του χρήστη. Έχοντας και άλλες επιλογές αποθηκευτικού χώρου όπως τα cookies, το local storage ή το session storage, καλύπτεται μόνον το κομμάτι του Cache API και τον αποθηκευτικό χώρο της κρυφής μνήμης (Cache Storage) αφού είναι σχεδιασμένος να συνεργάζεται με τους service workers και το Fetch API, όπως επίσης είναι σχεδιασμένος να χρησιμοποιεί τα promises.

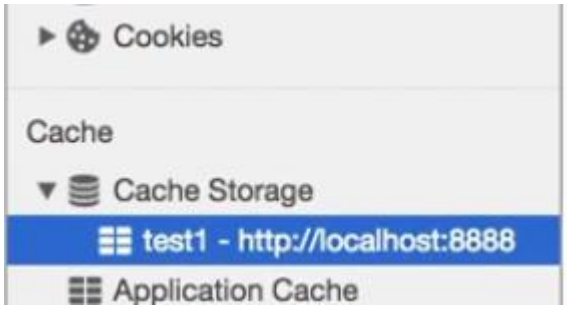
5.1 Cache Storage



Στα devtools του Chrome, επιλέγοντας το Application tab, παρατηρείται κάτω αριστερά το Cache Storage.

Στο main.js είναι απαραίτητο να γίνει έλεγχος εάν υποστηρίζεται το Cache API από τον browser. Εάν το API είναι διαθέσιμο υπάρχει πρόσβαση στο Cache Storage.

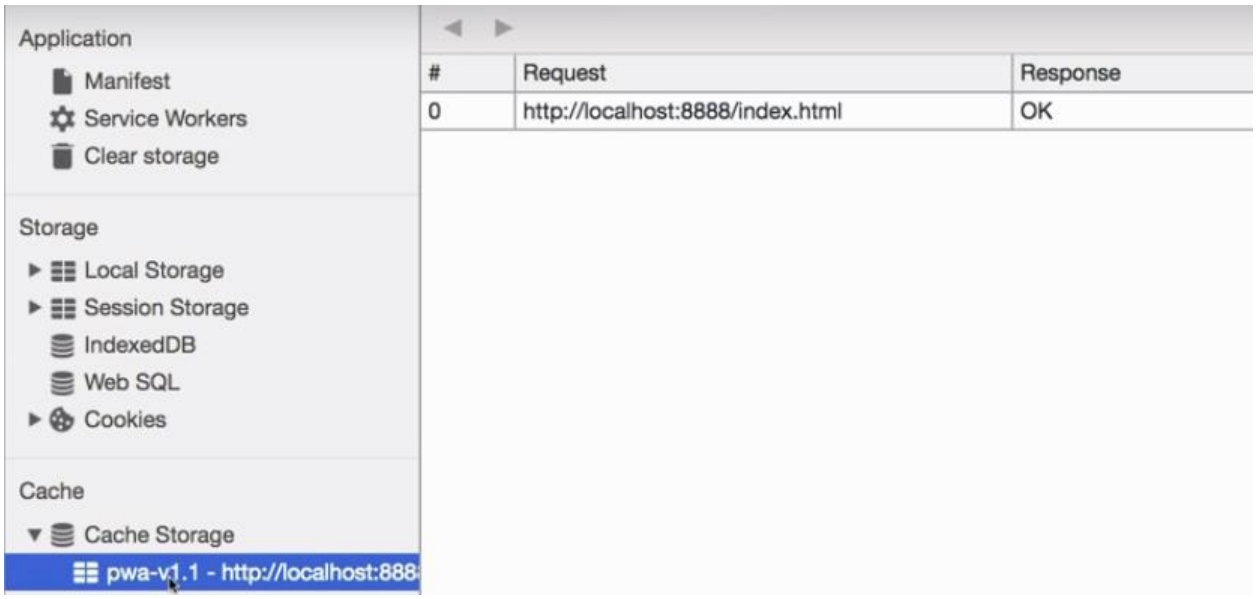
```
if (window.caches) { caches.open('test1') }
```



Η javascript ψάχνει για το Cache Storage ‘test1’ και εάν δεν υπάρχει το δημιουργεί.

Μπορεί να ξεκινήσει η χρήση της cache με το .then() μετά την open(), αφού η μέθοδος αυτή επιστρέφει ένα promise. Συγκριτικά με άλλες storage επιλογές που χρησιμοποιούν καταχωρήσεις των key τιμών ενός πίνακα από δεδομένα, το Cache API αποθηκεύει ένα response του server που αφορά κάποιο resource σε ένα request, χρησιμοποιώντας τα ίδια request και response objects που χρησιμοποιεί το Fetch API. Αυτό γίνεται μέσω της μεθόδου add().

```
caches.open(‘pwa-v1.1’).then( (cache) => {  cache.add(‘/index.html’); });
```



Στην πραγματικότητα η μέθοδος add() τρέχει μία fetch μέθοδο στο παρασκήνιο και αποθηκεύει το fetch response στο cache request. Το ίδιο μπορεί να γίνει για πολλαπλά αιτήματα χρησιμοποιώντας τη μέθοδο addAll(). Προσοχή, εάν αποτύχει ένα request η μέθοδος addAll() αποτυγχάνει.

```
cache.addAll([
    ‘/index.html’,
    ‘/style.css’,
    ‘main.js’
]);
```

#	Request	Response
0	http://localhost:8888/index.html	OK
1	http://localhost:8888/main.js	OK
2	http://localhost:8888/style.css	OK

Για να διαγραφεί ένα στοιχείο του Cache Storage δίνεται η εντολή delete().

```
cache.delete('/style.css');
```

Μπορούν επίσης να εμφανιστούν τα ίδια τα request objects της cache.

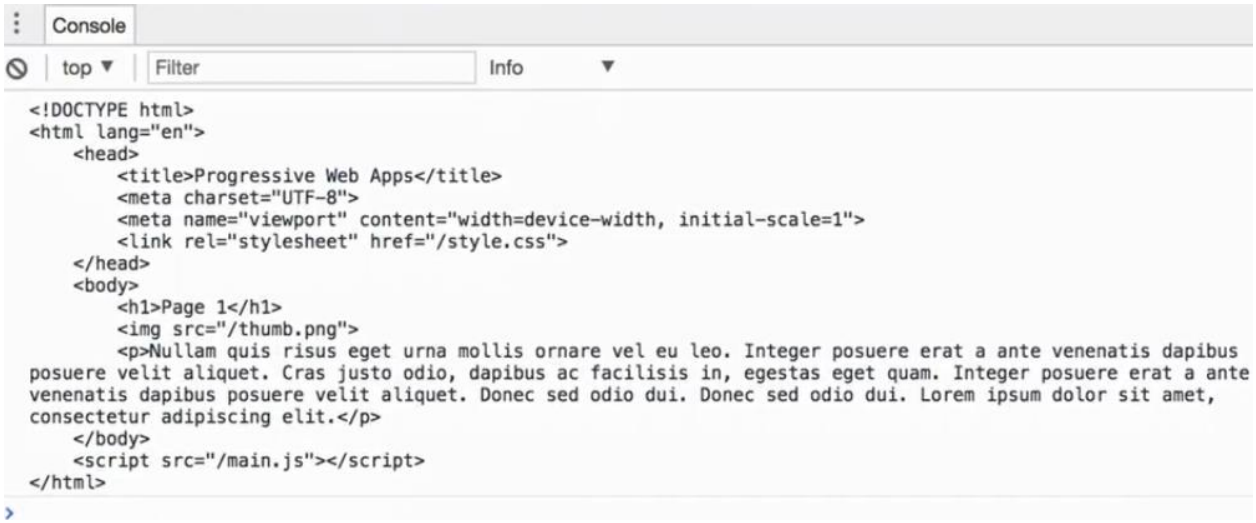
```
caches.keys().then(console.log);
```

Αυτές οι μέθοδοι επιστρέφουν promises όπως συμβαίνει και με το υπόλοιπο Cache API. Είναι χρήσιμη, λοιπόν, η αναμονή ώστε τα promises να γίνουν resolved. Για την ανάκτηση των cache καταχωρήσεων και για τη χρήση τους υπάρχει ο εξής τρόπος με την μέθοδο match():

```
cache.match('/index.html').then(console.log);
```

Για την εμφάνιση στην κονσόλα του response body χρησιμοποιείται το console.log() σε συνδυασμό με τη μέθοδος text() που μετατρέπει σε μορφή κειμένου το response.

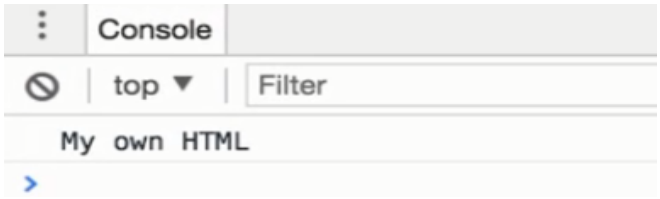
```
cache.match('/index.html').then( (res) => {

   res.text().then(console.log);

});
```



Προηγουμένως, αναφέρθηκε πως με τη μέθοδο add() εκτελείται μία fetch μέθοδος στο παρασκήνιο και αποθηκεύει το response της στο cache request. Αυτό συμβαίνει εσωτερικά στο API με την μέθοδο put(). Αυτή η μέθοδος μπορεί να εκτελεστεί κατευθείαν αφού βοηθάει στη χειραγώγηση του fetch response που αποθηκεύεται ως cache request ή για να αποφευχθούν άσκοπες κλήσεις της fetch. Για παράδειγμα:

```
cache.put('index.html', new Response('My own HTML'));
```

Η πρώτη παράμετρος είναι το request που θα αποθηκευτεί το response, η δεύτερη είναι το custom response object που φτιάχνεται μέσα από τον constructor Response(). Στην κονσόλα, το response body θα εμφανίζεται ως:



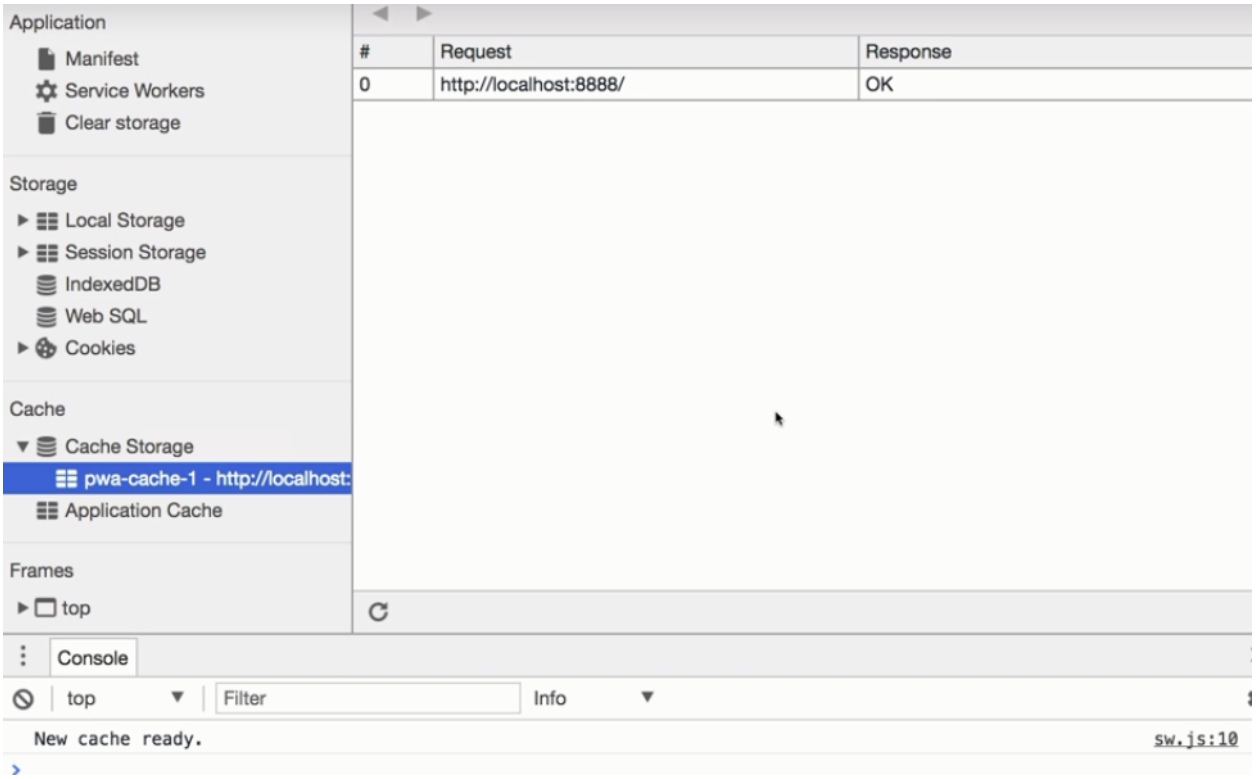
5.2 Caching Μέσα Στον Service Worker

Η *υλοποίηση της κρυφής μνήμης* σε ένα PWA γίνεται με το που τελειώσει το install του service worker. Εκείνη θα περιλαμβάνει όλα τα στατικά αρχεία που περιλαμβάνουν τη βασική δομή HTML, το στυλ, τις εικόνες και το βασικό JavaScript αρχείο της εφαρμογής. Αυτή η λογική πραγματοποιείται στο sw.js αρχείο με αυτόν τον τρόπο:

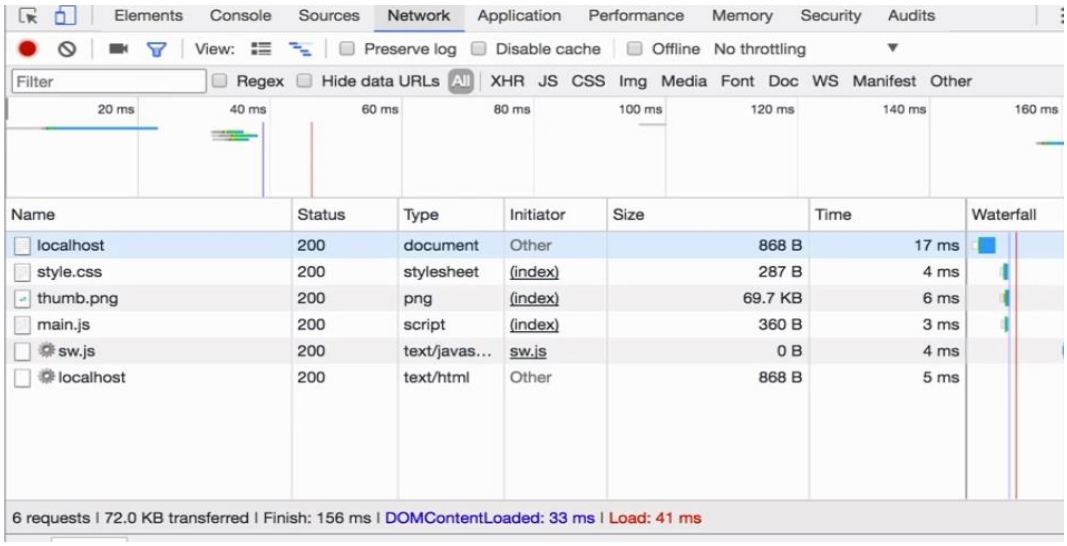
```
const pwaCache = 'pwa-cache-1';

self.addEventListener('install', (e) => {
```

```
    let cacheReady = caches.open(pwaCache).then((cache) => {
        console.log('New cache ready');
        return cache.add('/');
    });
    e.waitUntil(cacheReady);
});
```

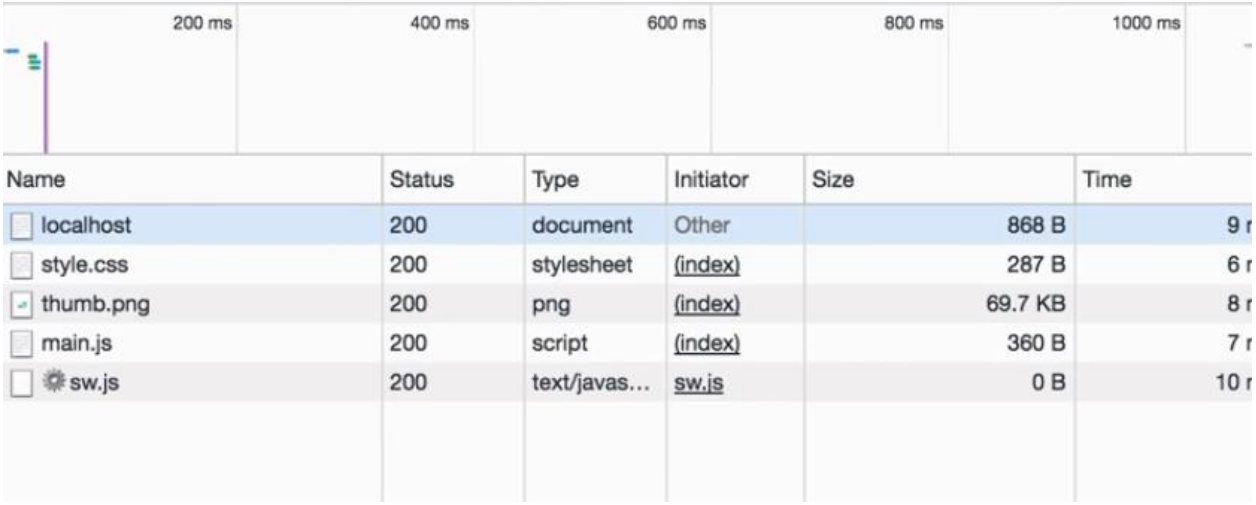
Κάνοντας inspect μέσα από το application tab του devtools παρατηρείται το μήνυμα στην κονσόλα και το root URL της εφαρμογής που έγινε add.



Ως υπενθύμιση, η μέθοδος add() εκτελεί μία background fetch μέθοδο για το request ώστε να αποθηκευτεί το σχετικό response. Πηγαίνοντας στο network tab, στη λίστα των στοιχείων που φορτώθηκαν, παρατηρείται στην αρχή το request στον localhost με το response να περιέχει το HTML αρχείο που φορτώνεται από τον browser. Στο τέλος της λίστας, παρατηρείται το request στον localhost που είναι πάλι το HTML αρχείο και που πραγματοποιήθηκε μετά το request του sw.js, το οποίο υποδεικνύεται από το εικονίδιο-γρανάζι στα αριστερά και που δεν έχει φορτωθεί μέσα από τον browser.



Μετά από το refresh της σελίδας, ο service worker έχει εγκατασταθεί και πλέον είναι active. Αυτό σημαίνει πως ο παραπάνω κώδικας δεν θα τρέξει αφού αφορά μόνο το install event.



Το index.html παραμένει στην cache. Σε αυτή την περίπτωση, το request για την HTML είναι σωστό να γίνεται μέσα από την cache και τον service worker παρά μέσα από request του browser. Για να γίνει αυτό θα πρέπει να στοχευθεί το fetch event μέσα από έναν eventListener.

```
self.addEventListener('fetch', (e) => {
   if (e.request.url === 'http://localhost:8888/') {
      let newRes = caches.open(pwaCache).then((cache) => {
         return cache.match(e.request);
```

```
        });
        e.respondWith(newRes);
    }
});
```

Κάνοντας update τον νέο service worker, μιας και τροποποιήθηκε το sw.js αρχείο, αρχικά γίνεται install αποθηκεύοντας στην cache το 'localhost/' και με ένα refresh στη σελίδα, ο service worker μπαίνει στην active κατάσταση. Παρατηρώντας τώρα το network tab, το response στο localhost request παραδόθηκε πρώτο και μέσα από τον service worker.

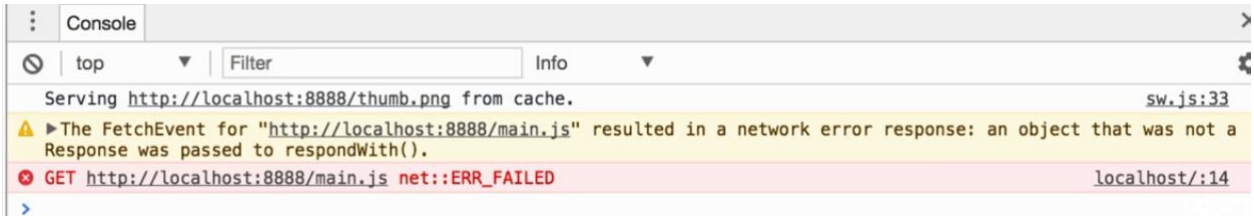
Name	Status	Type	Initiator	Size	Ti
localhost	200	document	Other	(from ServiceWorker)	
style.css	200	stylesheet	(index)	287 B	
thumb.png	200	png	(index)	69.7 KB	
main.js	200	script	(index)	360 B	
sw.js	200	text/javas...	sw.js	0 B	

Για να αναγκαστούν τα static resources να φορτώνονται κατά την offline εκτέλεση της εφαρμογής, θα προστεθούν στην cache και θα φορτώνονται από εκεί. Ένα παράδειγμα κώδικα στο sw.js θα είναι:

```
const pwaCache = 'pwa-cache-1';
self.addEventListener('install', (e) => {
   let cacheReady = caches.open(pwaCache).then((cache) => {
      console.log('New cache ready');
      return cache.addAll([
            '/',
            'style.css',
            'thumb.png'
          ]);
```

```
    });
    e.waitUntil(cacheReady);
});


self.addEventListener('fetch', (e) => {
    //skip for remote fetch
    if ( !e.request.url.match.location.origin  ) return;
    //serve local fetch from cache
    let newRes = caches.open(pwaCache).then((cache) => {
        return cache.match(e.request).then((res) => {
            //Check request was found in cache
            if (res) {
                //using ES6 string interpolation
                console.log(`Serving ${res.url} from cache.`);
                return res;
                    }
             });
      });
      e.respondWith(newRes);
});
```

Κάνοντας install τον νέο service worker και μετά το refresh της σελίδας παρατηρείται στην κονσόλα ένα error.



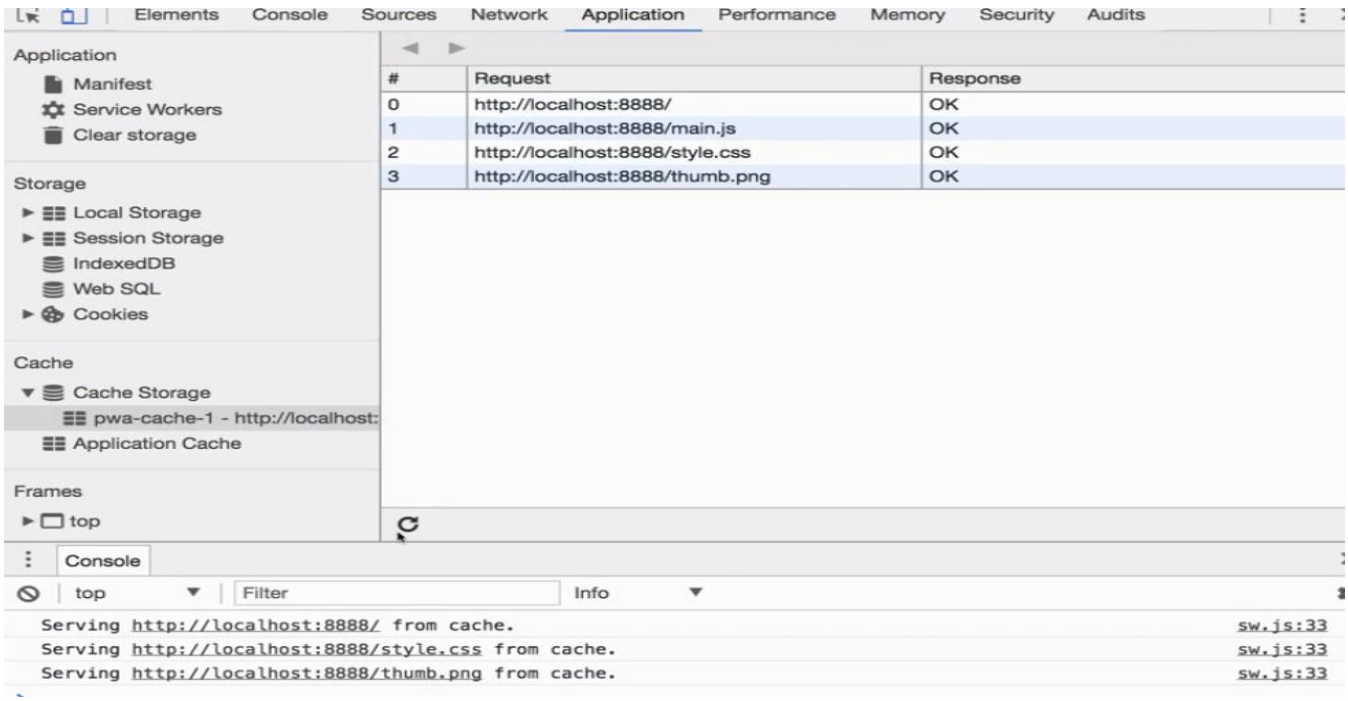
Σε μία τέτοια περίπτωση η σελίδα δεν θα φορτωθεί, αφού τη στιγμή εκείνη δε σερβίρεται το main.js στον client από το fetch event. Φυσικά ένας προγραμματιστής μπορεί να κάνει unregister τον service worker από τα devtools και να φορτωθεί το app όπως στην αρχή, όμως ο τελικός user που

θα χρησιμοποιήσει την εφαρμογή δε θα είναι ικανός να επιλύσει το πρόβλημα μόνος του και έτσι θα έχει μια εφαρμογή που δεν λειτουργεί. Ένα τέτοιο παράδειγμα, είναι μία πρόθεση να τονιστεί το **best practice** στο πως αποθηκεύονται requests για resources στην . Το πρόβλημα θα λυθεί έχοντας τους πόρους που λείπουν μέσα στον service worker και στην cache.

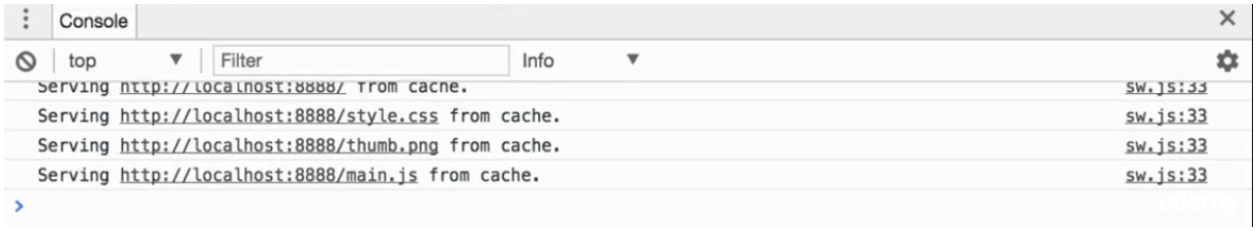
Εάν το response δεν βρέθηκε, θα πρέπει να γίνει fetch το πρωτότυπο request χειροκίνητα. Στον προηγούμενο κώδικα, μέσα στο “return cache.match(e.request).then((res) => { “, κάτω από το if (res){}, θα γίνει η προσθήκη της else:

```
}else{
    //Fetch on behalf of client and cache
    return fetch(e.request).then((fetchRes) => {
        cache.put(e.request, fetchRes.clone());
        return fetchRes;
    });
}
```

Κάνοντας unregister τον active service worker και κάνοντας επαναφόρτωση, ο νέος service worker εγκαθίσταται.

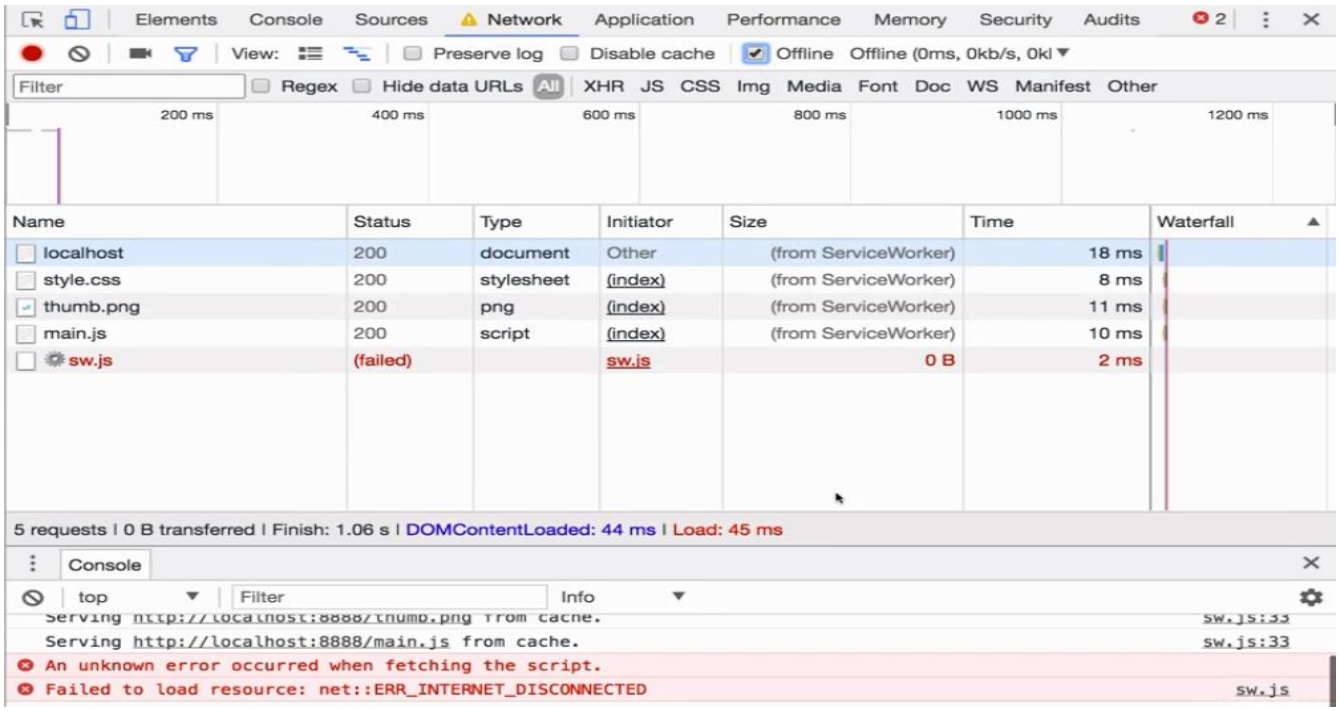


Στην κονσόλα παρατηρούνται τα 3 requests που σερβίρονται από την cache και μέσα στην cache παρατηρείται και το main.js. Με άλλο ένα refresh στη σελίδα μας, η κονσόλα θα εμφανίσει και το main.js, δηλώντας πως αυτή τη φορά σερβίρει το αρχείο μέσα από την cache.



Αυτό επιβεβαιώνεται και από το network tab.

Σε αυτό το σημείο σερβίρονται τα στατικά resources μιας εφαρμογής από την cache και όπως αναφέρθηκε, αυτή η διαδικασία κάνει διαθέσιμα τα resources αυτά στην offline εργασία. Πατώντας το offline checkbox στο network tab και επαναφορτώντας τη σελίδα, παρατηρείται:



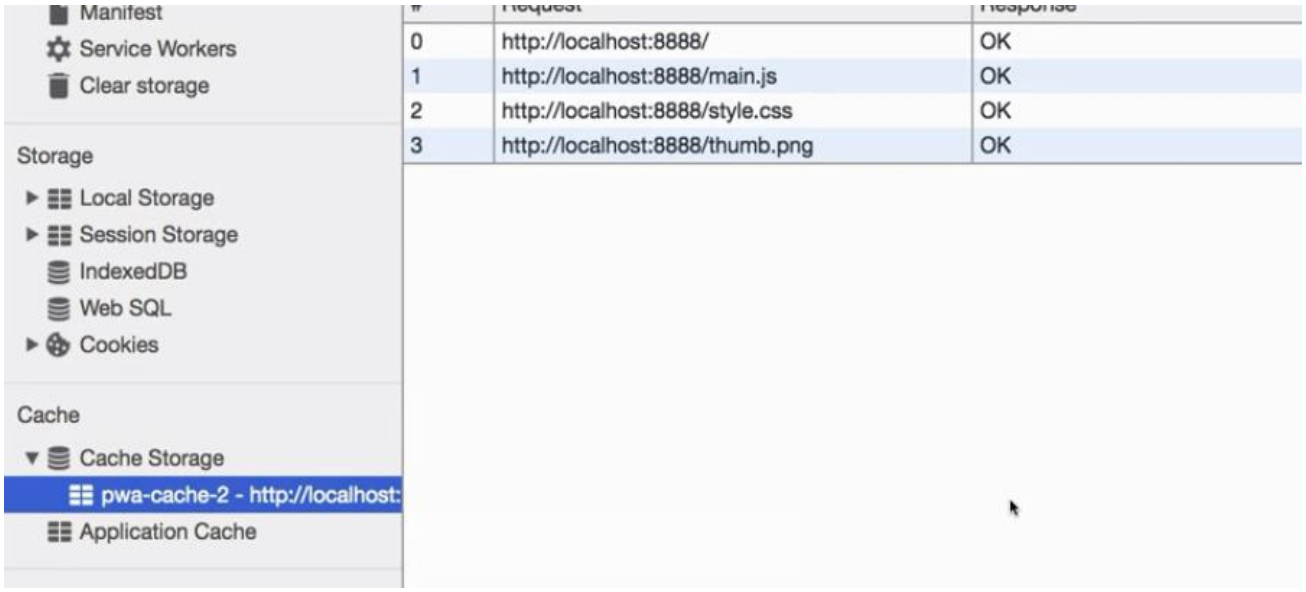
Το error που προκύπτει είναι εξαιτίας της αυτόματης ανανέωσης του service worker από τον browser κατά το refresh της σελίδας. Παράλληλα όμως, τα στατικά αρχεία φορτώνονται.

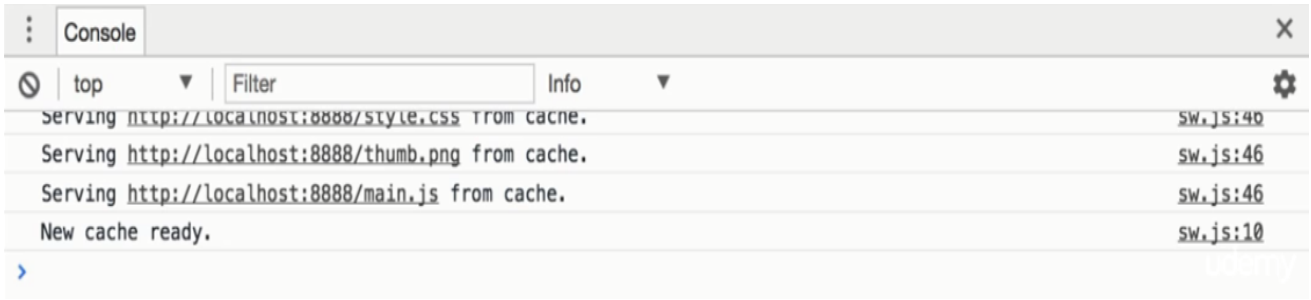
Το όνομα που δώθηκε στην cache που φιλοξενείται από το Cache Storage αντιπροσωπεύει την έκδοση της. Έστω ότι αφαιρείται ένα αρχείο από την εφαρμογή και κατ' επέκταση δεν χρειάζεται στην cache πια. Αντί της ενασχόλησης ξεχωριστά με το αρχείο, θα γίνει ενασχόληση αποκλειστικά της cache μέσα από τον service worker. Αυτό σημαίνει ότι όποτε γίνουν αλλαγές στον service worker θα ήταν σωστό να αλλάζει το όνομα της cache, η τιμή δηλαδή της pwaCache μεταβλητής. Εκείνη με τη σειρά της, θα δημιουργηθεί με το install του νέου service worker αλλά δε θα σβηστεί η παλιά. Αυτό θα μπορούσε να γίνει χειροκίνητα στο Cache Storage κάνοντας δεξί κλικ στην προηγούμενη cache και επιλέγοντας το remove, σβήνοντας έτσι την ανεπιθύμητη cache. Ο χρήστης όπως προαναφέρθηκε όμως, δεν θα είναι σε θέση να το κάνει αυτό και έτσι θα πρέπει να αυτοματοποιηθεί η διαδικασία αυτή της εκκαθάρισης.

Κατά το βήμα ενεργοποίησης του νέου service worker λοιπόν, θα τσεκάρονται τα στοιχεία όλων των προηγούμενων cache αποθηκευτικών χώρων και εάν δεν συμβαδίζουν με εκείνα της καινούριας cache, θα αφαιρούνται. Στο sw.js αρχείο, το όνομα της cache αλλάζει σε "pwa-cache-2", στο addAll() θα προστεθεί και το main.js και επίσης θα προστεθεί στον κώδικα το εξής κομμάτι:

```
self.addEventListener('activate', (e) => {
    let cacheCleaned = caches.keys().then((keys) => {
        keys.forEach(key) => {
            if( key!== pwaCache) {
                return caches.delete(key);
            }
        });
    });
    e.waitUntil(cacheCleaned);
});
```

Όταν εγκατασταθεί ο νέος service worker, παρατηρείται πως η προηγούμενη έκδοση "cache pwa-cache-1" αφαιρέθηκε και στη θέση της φορτώθηκε η νέα "pwa-cache-2" με όλα τα στατικά resources μέσα.


Manifest
Service Workers
Clear storage
Storage
Local Storage
Session Storage
IndexedDB
Web SQL
Cookies
Cache
Cache Storage
pwa-cache-2 - http://localhost:
Application Cache
0 http://localhost:8888/ OK
1 http://localhost:8888/main.js OK
2 http://localhost:8888/style.css OK
3 http://localhost:8888/thumb.png OK


Console
top
Filter
Info
Serving http://localhost:8888/style.css from cache. sw.js:40
Serving http://localhost:8888/thumb.png from cache. sw.js:46
Serving http://localhost:8888/main.js from cache. sw.js:46
New cache ready. sw.js:10

ΚΕΦΑΛΑΙΟ 6

Caching Στρατηγικές

Αφού αναφέρθηκε το πως εισάγεται η λογική της κρυφής μνήμης στον service worker, γίνεται αναφορά σε κάποιες από τις κοινές caching στρατηγικές που χρησιμοποιεί ένα PWA. Ένα PWA δεν χρησιμοποιεί μία μόνο στρατηγική. Η κάθε μία στρατηγική αφορά ένα διαφορετικό σενάριο που εξαρτάται από τη φύση των resources. Για παράδειγμα, είναι απαραίτητο όλα τα resources που απαρτίζουν μία εφαρμογή να χρησιμοποιηθούν σε offline λειτουργία; Δυναμικά δεδομένα όπως για παράδειγμα ένα προϊόν μέσα στο καλάθι πριν την παραγγελία, που ένας user διαμορφώνει και έχει προσθέσει ή αφαιρέσει επιλογές καθορίζοντας την τελική του τιμή, είναι δυνατόν να εμφανίζεται χωρίς σύνδεση στο διαδίκτυο, όμως δεν είναι δυνατόν να επιτραπεί συνέχεια στην παραγγελία αφου η επικοινωνία με τον server έχει διακοπεί. Σε εργασία χωρίς σύνδεση και αφού είναι αδύνατη η αλληλεπίδραση με τον server, είναι αξιοσημείωτο πως ο χρήστης θα πρέπει να ενημερώνεται πως δεν μπορεί να προχωρήσει στην επιθυμητή λειτουργία χωρίς σύνδεση στο διαδίκτυο. Οπότε, κάποια resources όπως η HTML σελίδα που αφορά την ολοκλήρωση της παραγγελίας και όποια άλλα resources την απαρτίζει είναι άχρηστα στην offline λειτουργία. Resources που αφορούν στατικά δεδομένα, όπως ένα css αρχείο, είναι τέλεια για να φορτώνονται κατευθείαν από την cache, ανεξαρτήτως σύνδεσης.

Είναι αποδεκτό να εμφανίζεται ένα περιεχόμενο μόνο στην πιο πρόσφατη του έκδοση;

Δεδομένα για παράδειγμα, όπως τα σχόλια χρηστών για ένα προϊόν θα παρουσιαστούν στην πιο πρόσφατη τους έκδοση που θα φορτωθούν από την cache όσο αυτή ανανεώνεται εφόσον το application έχει πρόσβαση στο δίκτυο. Στην offline λειτουργία όμως, θα ήταν καλύτερα το comment section να έχει κάποια στοιχεία και να ενημερώνεται ο user πως πρέπει να συνδεθεί για να εμφανιστούν τα νεότερα. Το να έχουν παρουσιαστεί δηλαδή, έστω κάποια παλαιότερα σχόλια στο section, είναι προτιμότερο από το section να είναι άδειο.

Τα resources που φορτώνονται από την cache προσφέρουν σημαντικό *όφελος απόδοσης* σε σύγκριση με εκείνα που φορτώνονται από το δίκτυο και κάποιες φορές *η απόδοση είναι πιο σημαντική από το να υπάρχουν διαθέσιμα τα πιο πρόσφατα δεδομένα*. Διάφοροι παράγοντες πρέπει να ληφθούν υπ’όψιν, όπως το data cost σε αναλογία με την πολυπλοκότητα του app.

Για τα παρακάτω στρατηγικά σενάρια, ο κώδικας στο sw.js αρχείο είναι ο εξής:

```
// Service Worker

// Cache name

const pwaCache = 'pwa-cache-1';
```

```
// Static assets to cache on install
const staticCache = [ '/', 'index.html', '/style.css', '/main.js' ];
// SW install and cache static assets
self.addEventListener('install', (e) => {
    e.waitUntil(
      caches.open(pwaCache)
        .then( cache => cache.addAll(staticCache) )
    );
});
// SW Activate and cache cleanup
self.addEventListener('activate', (e) => {
    let cacheCleaned = caches.keys().then((keys) => {
        keys.forEach( (key) => {
            if (key !== pwaCache) return caches.delete(key);
        });
    });
    e.waitUntil(cacheCleaned);
});
```

Αναφέρεται παρακάτω ξεχωριστά η κάθε στρατηγική, προσθέτοντας κάθε φορά κάτω από τον παραπάνω κώδικα διαφορετικό fetch event που αφορά την εκάστοτε στρατηγική, αφαιρώντας το προηγούμενο εφόσον υπάρχει.

Στρατηγικές:

- **Cache Only, Static Assets**

  Η πιο απλή στρατηγική είναι η μεταφορά των στατικών πόρων στην cache. Προσοχή, όπως ο user μπορεί να διαγράψει την cache έτσι μπορεί και το λειτουργικό του σύστημα. Δεν μπορει πάντα να είναι έγκυρη η διαθεσιμότητα των καταχωρήσεων στην cache.

```
self.addEventListener('fetch', (e) => {
    e.respondWith(caches.match(e.request));
```

```
});
```

- **Cache With Network Fallback**

  Αυτή η στρατηγική ουσιαστικά είναι η ίδια με την πρώτη αλλά με την διαφορά πως σε περίπτωση που η cache για κάποιο λόγο χάσει δεδομένα, θα τα ανακτήσει από το Διαδίκτυο. Είναι ένας πιο ισχυρός και αξιόπιστος τρόπος για την παραμονή στατικών πόρων στην cache.

```
self.addEventListener('fetch', (e) => {
  e.respondWith(
    caches.match(e.request).then( (res) => {
      if(res) return res;

      // Fallback
      return fetch(e.request).then( (newRes) => {
        // Cache fetched response
        caches.open(pwaCache).then(
       cache => cache.put(e.request, newRes)
        );
        return newRes.clone();
      })
    })
});
```

- **Network With Cache Fallback**

  Σε περίπτωση που δεν μπορεί να φορτωθεί η πιο πρόσφατη έκδοση των πόρων από το Διαδίκτυο, τρέχει στην cache ως εφεδρικό σχέδιο.Μια τέτοια επιλογή είναι καλή για να παρουσιαστούν δεδομένα, που ιδανικά θα είναι up-to-date, αλλά που ακόμη έχουν αξία και σε παλαιότερη εκδόση. Δεδομένα όπως τα σχόλια χρηστών για ένα προϊόν ή μια υπηρεσία, θα ταίριαζε να χρησιμοποιηθούν με αυτή τη στρατηγική.

```
self.addEventListener('fetch', (e) => {
  e.respondWith(
    fetch(e.request).then( (res) => {

      // Cache latest version
      caches.open(pwaCache).then(
        cache => cache.put(e.request, res)
      );
      return res.clone();

    // Fallback to cache
   }).catch( err => caches.match(e.request) )
  );
});
```

- **Cache With Network Update**

  Λύνοντας προβλήματα χαμηλής συνδεσιμότητας, αυτή η στρατηγική τραβάει περιεχόμενο από την cache, το σερβίρει στον user και μετά ανανεώνει την cache στο παρασκήνιο. Με αυτό τον τρόπο, στο επόμενο reload της σελίδας, η cache θα είναι όσο γίνεται up-to-date. Αυτή η στρατηγική δεν είναι καλή για real-time δεδομένα αλλά όπου ένας τέτοιος συμβιβασμός είναι αποδεκτός, είναι καλή εάν ληφθεί υπ όψιν η απόδοση.

```
self.addEventListener('fetch', (e) => {
    e.respondWith(
      caches.open(pwaCache).then( (cache) => {
        // Return from cache
        return cache.match(e.request).then( (res) => {

          // Update
```

```
          let updatedRes = fetch(e.request).then( (newRes) => {
            // Cache new response
            cache.put(e.request, newRes.clone());
            return newRes;
          });

          return res || updatedRes;
        })
      })
    );
});
```

- **Cache & Network Race**

  Η πιο εξειδικευμένη στρατηγική που καλύπτει πολλά σενάρια, προτείνει να αποσταλεί στο δίκτυο ένα request και παράλληλα να αποσταλεί ένα request στην cache. Στη συνέχεια, αναγνωρίζεται ως ‘πρώτο’ αυτό που έστειλε πιο γρήγορα response και τα επόμενα requests γίνεται αυτόματα reject. Η cache φαινομενικά πάντοτε είναι η πρώτη που επιστρέφει response, όμως σε κάποιες συσκευές η πρόσβαση στο δίσκο μπορεί να αποδειχθεί πιο αργή από την πρόσβαση στο δίκτυο.

```
self.addEventListener('fetch', (e) => {
    let firstResponse = new Promise((resolve, reject) => {
      // Track rejections
      let firstRejectionReceived = false;
      let rejectOnce = () => {
        if (firstRejectionReceived) {
            reject('No response received.')
         } else {
          firstRejectionReceived = true;
```

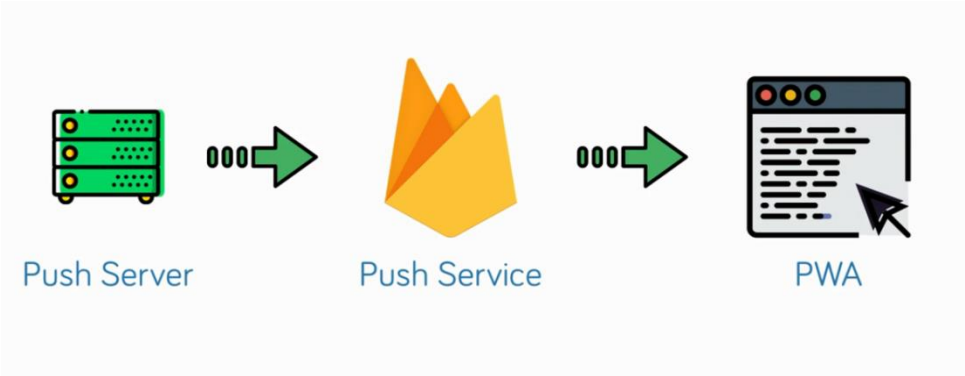
```
      }
    };
    // Try Network
    fetch(e.request).then( (res) => {
      // Check res ok
      res.ok ? resolve(res) : rejectOnce();
    }).catch(rejectOnce);
    // Try Cache
    caches.match(e.request).then( (res) => {
      // Check cache found
      res ? resolve(res) : rejectOnce();
    }).catch(rejectOnce);
  });
  e.respondWith(firstResponse);
});
```

ΚΕΦΑΛΑΙΟ 7

Κατασκευάζοντας Έναν Push Διακομιστή

Η υπηρεσία Push είναι διαθέσιμη από τον browser και λειτουργεί ως υπηρεσία μεσολάβησης. Στον Chrome για παράδειγμα, η υπηρεσία Push που προσφέρεται θα είναι η υπηρεσία Google firebase Cloud messaging η οποία διαχειρίζεται από τον browser στο παρασκήνιο και δεν απαιτεί από τον χρήστη δημιουργία λογαριασμού. Κάποιες Push υπηρεσίες από εξωτερικό πάροχο είναι η pushcrew, η OneSignal και η Amazon Simple Notification Service, όπως επίσης υπάρχουν αντίστοιχες για PHP, Python και Java.

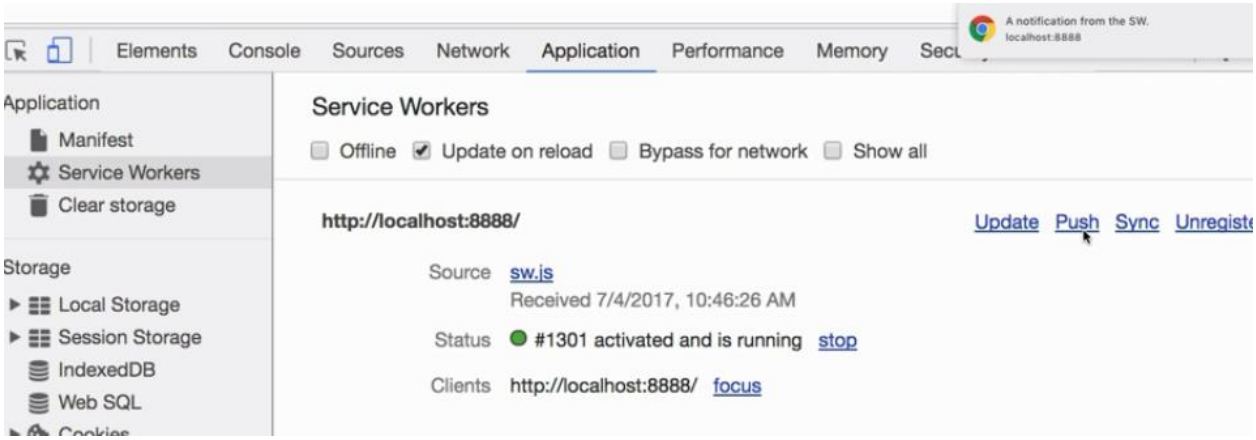
Η πρόσβαση σε πιο εξειδικευμένα χαρακτηριστικά όταν ένας ιστότοπος ή μία εφαρμογή κάνει εγγραφή σε μία Push υπηρεσία απαιτεί τη διανομή κλειδιών αυθεντικοποίησης από την υπηρεσία αυτή. Έπειτα, ένας Push server που γνωρίζει τα διαπιστευτήρια της εφαρμογής, μπορεί να στείλει κωδικοποιημένες ειδοποιήσεις (payloads) στην υπηρεσία και εκείνη να τις προωθήσει στην εφαρμογή. Η κωδικοποίηση μεταξύ της εφαρμογής και του push server θα γίνει μέσω ενός private και ενός public key και έτσι η εφαρμογή θα μπορεί να αποκρυπτογραφήσει τα payloads της υπηρεσίας που στέλνονται από τον Push Server.



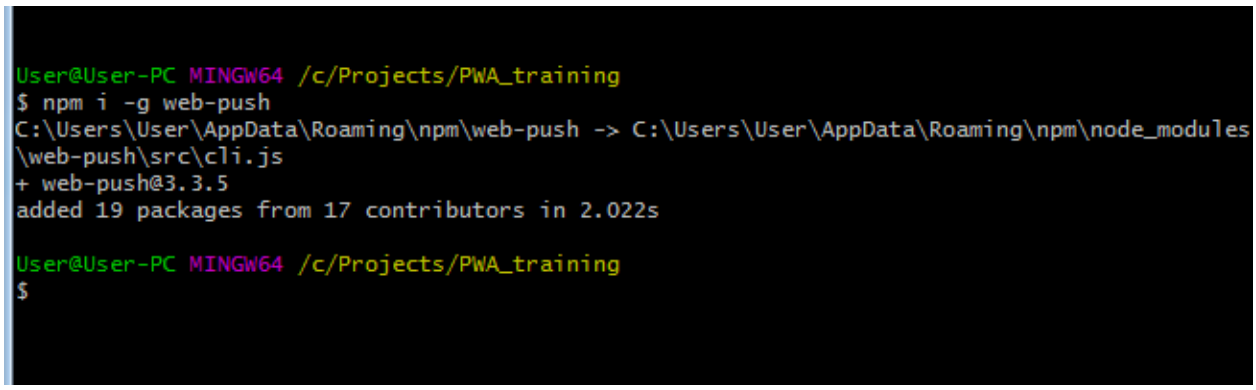
Όποτε ο Push server προωθεί ειδοποίηση στην εφαρμογή, ο service worker θα πρέπει με τη σειρά του να την προωθεί - παρουσιάζει στο χρήστη. Προστίθεται στο sw.js ο εξής κώδικα:

```
//Send notifications on push
self.addEventListener('push', (e) => {
    let n = self.registration.showNotification(
        'A notification from SW.');
    e.waitUntil(n);
});
```

Όταν εγκατασταθεί ο νέος service worker μετά από refresh της σελίδας, μέσα στο application tab των devtools του Chrome, επιλέγοντας την “Service Workers” επιλογή και εάν γίνει κλικ στο Push σύνδεσμο, θα γίνει triggered μία προσομοίωση του push event και θα εμφανιστεί το notification από τον service worker.



Για τη δημιουργία του push server χρησιμοποιείται το web push module σε συνδυασμό με το node.js. Το module απαιτείται να είναι εγκατεστημένο. Εγκαθίσταται μέσω του terminal (χρησιμοποιείται το Git Bash), χρησιμοποιώντας τον NPM (Node Package Manager). Η εντολή για την εγκατάσταση είναι η “npm i web-push”. Για global install, γίνεται προσθήκη του -g στην εντολή εγκατάστασης.



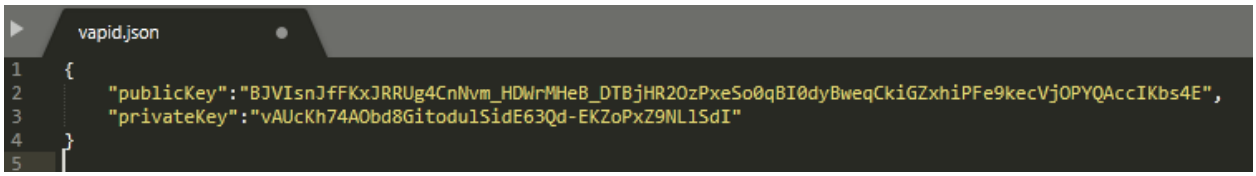
Θα χρειαστεί για την εφαρμογή ένα νέο αρχείο js που ονομάζεται push-server.js και μέσα θα καλείται το web module.

```
const webpush = require('web-push');
```

Όπως αναφέρθηκε, θα χρειαστούν κλειδιά αυθεντικοποίησης ώστε να αναγνωριστεί ο push server από την push υπηρεσία. Από τον terminal εκτελείται η εντολη:

```
web-push generate-vapid-keys --json > vapid.json
```

Με τον παραπάνω κώδικα αποθηκεύονται τα keys σε ένα json αρχείο που δημιουργείται. Αφού τα keys είναι στατικά καλούνται μέσα από τον push server χρησιμοποιώντας το json σαν javascript.



Έπειτα, θα διαμορφωθεί ο server να χρησιμοποιεί αυτά τα κλειδιά.

```
const vapid = require('./vapid.json');

webpush.setVapidDetails(

    'mailto:stefanoskokkalhs@gmail.com',

    vapid.publicKey,

    vapid.privateKey

);

const pushSubscription = {

    endpoint: '',

    keys: {

        auth: '',

        p256dh: ''

       }

};

webpush.sendNotification(pushSubscription, 'A notification from the push
server');

console.log('Push send to client');
```

Για να ληφθούν οι λεπτομέρειες του subscription, στο main.js μέσα στο register του service worker προστίθεται ο εξής κώδικα:

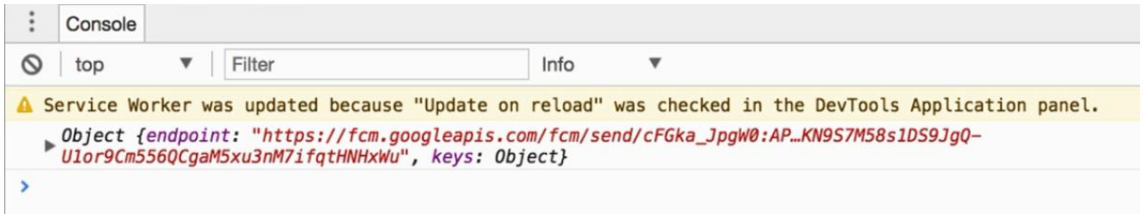
```
//Server Public Key

//Copy Paste public key from vapid.json
```

```
let pubKey =
"BJVIsnJfFKxJRRUg4CnNvm_HDWrMHeB_DTBjHR2OzPxeSo0qBI0dyBweqCkiGZxhiPFe9kecVjOPYQA
ccIKbs4E";

registration.pushManager.getSubscription().then((sub) => {

    // If subscription found, return

    if (sub) return sub;

    let key = urlBase64ToUint8Array(pubKey);

    // Subscribe

    return registration.pushManager.subscribe({

        userVisibleOnly: true,

        applicationServerKey: key

    });

}).then( sub => sub.toJSON() )

    .then(console.log)

    .catch(console.log);
```

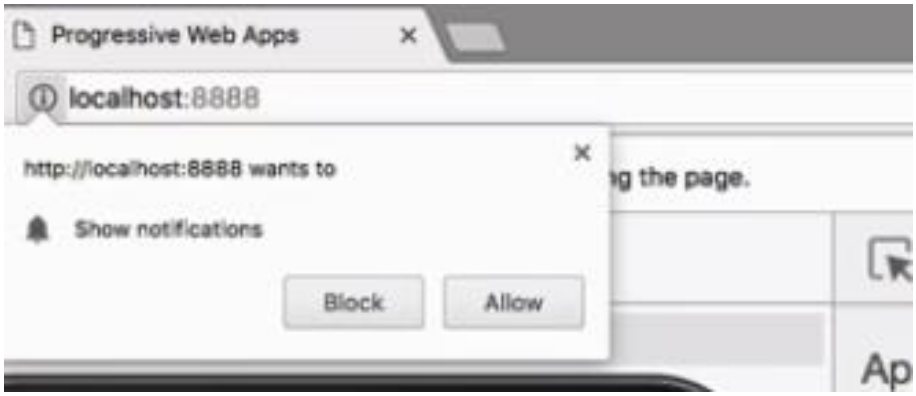
Στο documentation του module (https://www.npmjs.com/package/web-push) αναφέρεται πως για να χρησιμοποιηθούν τα vapid keys μέσα από την εφαρμογή στο subscribe() θα πρέπει να μετατραπούν από base64 string σε Uint8Array. Πάνω από τον παραπάνω κώδικα και πάλι μέσα στο register του service worker, για να λειτουργήσει το urlBase64ToUint8Array(), θα προστεθεί ο κώδικας της σελίδας:

```
function urlBase64ToUint8Array(base64String) {

      const padding = '='.repeat((4 - base64String.length % 4) % 4);

      const base64 = (base64String + padding).replace(/-/g, '+')

      .replace(/-/g, '/');

      const rawData = window.atob(base64);

      const outputArray = new Uint8Array(rawData.length);

      for (let i = 0; i < rawData.length; ++i) {

       outputArray[i] = rawData.charCodeAt(i);

      }

      return outputArray;

}
```

Επαναφορτώνοντας την εφαρμογή, παρατηρείται στην κονσόλα του browser το subscription object.



Εάν δεν έχει δωθεί permission granted στα push notifications, ο browser θα ρωτήσει εάν επιτρέπονται οι ειδοποιήσεις.



ΚΕΦΑΛΑΙΟ 8

Native Αισθητική

Τα native apps αναπτύσσονται για μια συγκεκριμένη πλατφόρμα χρησιμοποιώντας άλλη γλώσσα προγραμματισμού για την κάθε πλατφόρμα. Οι εφαρμογές σε iOS πλατφόρμες για παράδειγμα, είναι ανεπτυγμένες σε Objective-C και Swift, οι εφαρμογές σε Android είναι σε Java ή Kotlin και οι εφαρμογές Windows σε C++. Μπορούν να αξιοποιήσουν τις ενσωματωμένες λειτουργίες της συσκευής για την οποία προορίζονται όπως το GPS, τον ανιχνευτή κίνησης ή την κάμερα. Τέλος, ένα native app έχει ένα εικονίδιο εκκίνησης (**launch icon**) στο homescreen της συσκευής που δηλώνει πως η εφαρμογή είναι εγκατεστημένη και είναι έτοιμη να χρησιμοποιηθεί όταν επιλέγεται, καθώς και το **splash screen** κατά την εκκίνηση της που εμφανίζεται μέχρι να φορτωθεί.

Τα Progressive Web Apps προσφέρουν τη δυνατότητα μετατροπής ενός web app, που είναι προσβάσιμο από τον browser, σε ένα native-like app, που μπορεί να εγκατασταθεί σε μία mobile συσκευή με android ή iOS λειτουργικό και που θα είναι προσβάσιμο από το home screen.

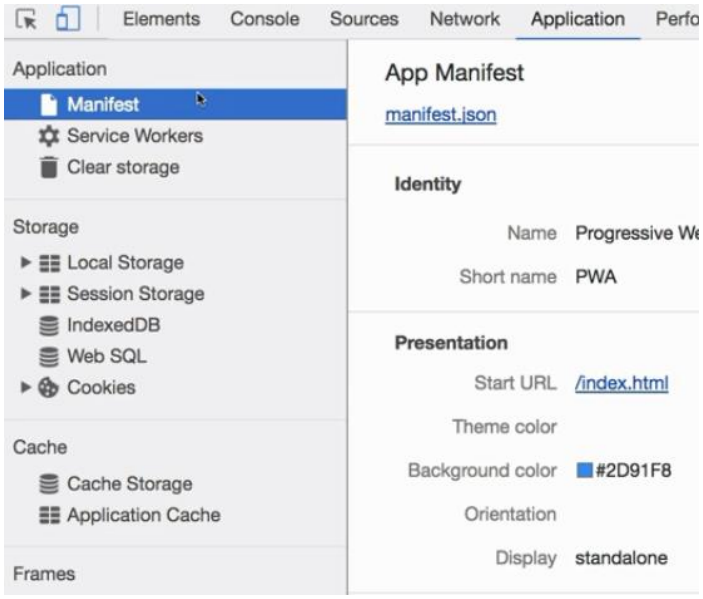
8.1 App Manifest

Το Web App Manifest επιτρέπει στην εφαρμογή ουσιαστικά να φύγει από τον browser και να γίνει μία αυτόνομη εφαρμογή στη συσκευή του χρήστη. Σε αντίθεση με το caching και τον service worker, το web app manifest είναι ένα αρκετά ξεκάθαρο JSON αρχείο που παρέχει στον browser πληροφορίες για την εφαρμογή, και δεν απαιτεί κώδικα. Το JSON αρχείο δηλώνεται στην HTML, κάτω από τις δηλώσεις των υπόλοιπων πόρων μέσα στο <head>, καθώς επίσης προτείνεται η δήλωση του στο array που αφορά τα static resources και που καλείται στον service worker για να αποθηκευτεί στην cache. Μπορεί να γίνει generate μέσα από έναν online manifest generator ή να συνταχθεί χειροκίνητα. Παράδειγμα του JSON αρχείου στην εικόνα δεξιά.

manifest.json

```
{
  "name": "Progressive Web App",
  "short_name": "PWA",
  "start_url": "/index.html",
  "display": "standalone",
  "background_color": "#2D91F8",
  "description": "",
  "icons": [
    {
      "src": "/icons/icon-72.png",
      "sizes": "72x72",
      "type": "image/png"
    },
    {
      "src": "/icons/icon-96.png",
      "sizes": "96x96",
      "type": "image/png"
    },
    {
      "src": "/icons/icon-128.png",
      "sizes": "128x128",
      "type": "image/png"
    },
    {
      "src": "/icons/icon-144.png",
      "sizes": "144x144",
      "type": "image/png"
    },
    {
      "src": "/icons/icon-152.png",
      "sizes": "152x152",
      "type": "image/png"
    },
    {
      "src": "/icons/icon-192.png",
      "sizes": "192x192",
      "type": "image/png"
    }
  ]
}
```

Όταν το app manifest είναι σωστά εγκατεστημένο στην εφαρμογή, μπορεί να παρατηρηθεί μία περίληψη του στα devtools, πηγαίνοντας στο application tab και επιλέγοντας το manifest πάνω από τον service worker.

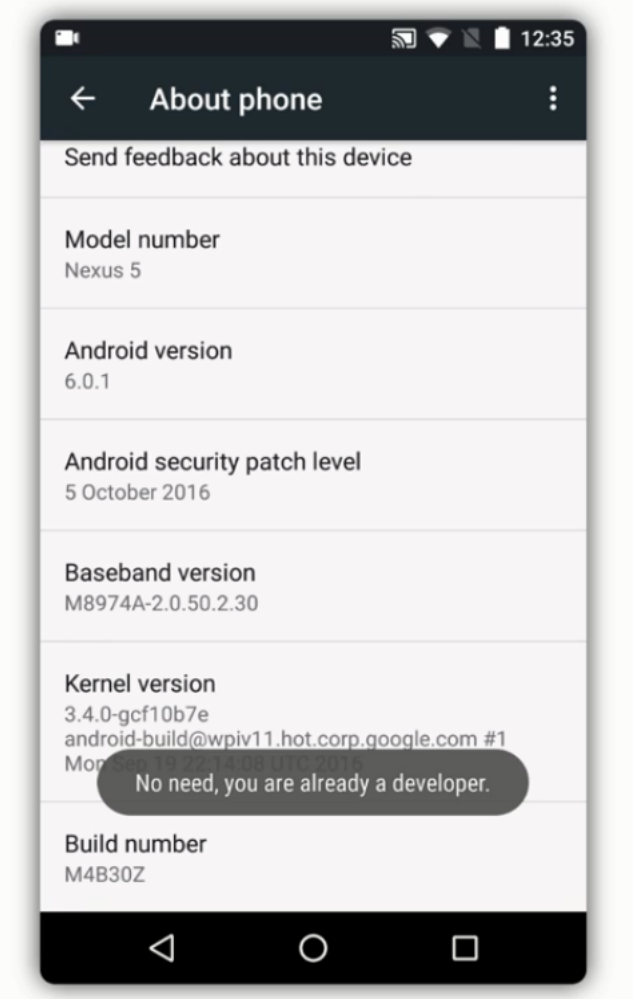


8.2 Εκτελώντας την Εφαρμογή Σε Κινητές Συσκευές Από Τον Localhost

Ένα έτοιμο PWA που **έχει πιστοποιηθεί και τρέχει στο πρωτόκολλο HTTPS**, μπορεί να τρέξει μέσα από μία αληθινή κινητή συσκευή και να προβληθούν οι δυνατότητες του εγκατεστημένου μας App Manifest. Ευτυχώς όμως, υπάρχει και η δυνατότητα μέσα από τον Chrome του υπολογιστή, να εντοπιστεί μία mobile συσκευή συνδεδεμένη με ένα καλώδιο USB και να τρέξει μια εφαρμογή από τον localhost.

Τι χρειάζεται η διαδικασία του USB Debugging:

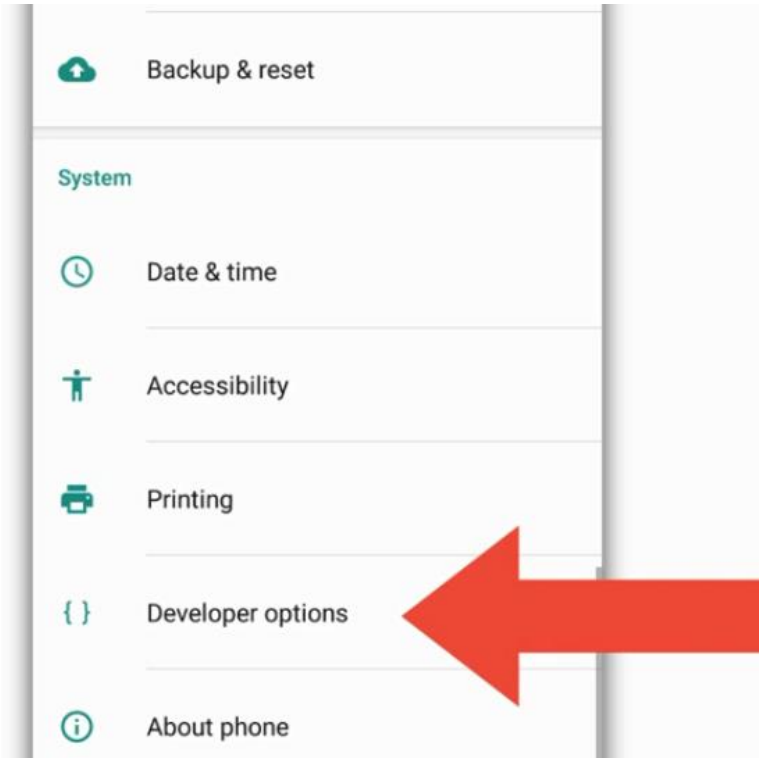
- Chrome 47 ή μεταγενέστερη έκδοση.
- Web Server για Chrome ή έναν active localhost.
- Μια συσκευή Android που εκτελείται έκδοση 4.0 ή νεότερη και Chrome 40 ή μεταγενέστερη.
- Ένα καλώδιο USB για τη σύνδεση του υπολογιστή στη συσκευή.



Για τον ίδιο σκοπό και χωρίς τη χρήση κάποιου browser, υπάρχουν και επιλογές προσομοίωσης του Android συστήματος στο desktop όπως εκείνη που έρχεται μαζί με το Android SDK, ή κάποιος εναλλακτικός προσομοιωτής όπως το GENYMOTION, καθώς και του λειτουργικού iOS μέσω του προσομοιωτή που έρχεται με το Xcode για MAC χωρίς να συνδυαστεί συσκευή.

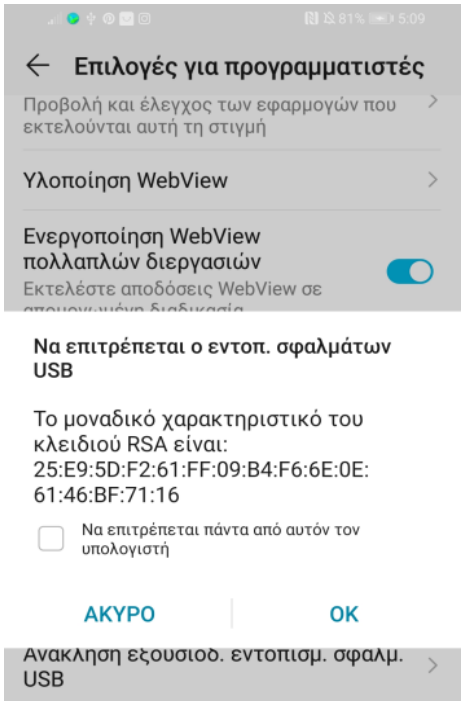
Για να επιτευχθεί η σύνδεση του Chrome με μία αληθινή συσκευή Android, θα πρέπει να ενεργοποιηθούν τα ‘Developer Options/Επιλογές Προγραμματιστή’ στην συσκευή.

Στα settings του android, κάνοντας scroll στο τέλος των επιλογών, θα παρατηρηθούν τα ‘developer options’.

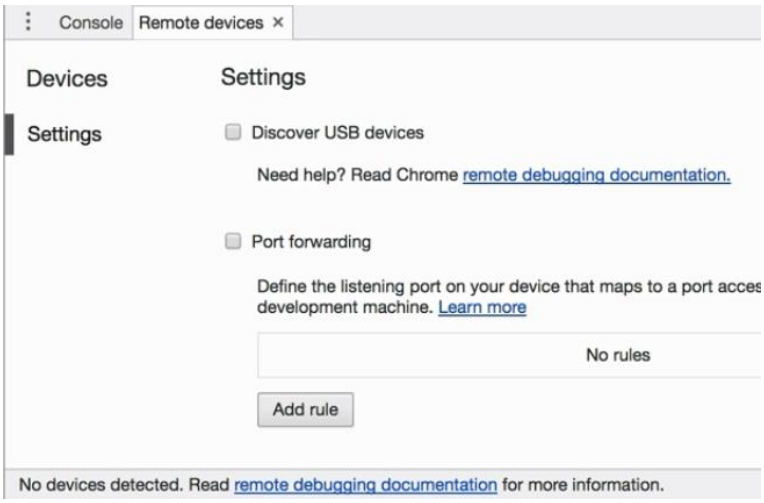


Αν δεν είναι εμφανισμένη η επιλογή θα πρέπει να πρώτα να ενεργοποιηθεί. Πρέπει να γίνει πλοήγηση στα settings της συσκευής, στην επιλογή ‘About phone’ και να γίνει αναζήτηση του ‘Build number’ της συσκευής. Πατώντας 7 φορές πάνω του ενεργοποιείται η επιλογή ‘Developer options’ και είναι ορατή πλέον στα settings.

Έπειτα, πραγματοποιείται η σύνδεση μέσω του καλωδίου USB μεταξύ του desktop και της mobile συσκευής. Η επιλογή που χρειάζεται να ενεργοποιηθεί στα developer options είναι το ‘USB debugging’. Μόλις ενεργοποιηθεί και αφού είναι ενεργοποιημένη η λειτουργία από τον browser του desktop θα εμφανιστεί ένα παράθυρο δείχνοντας ένα μήνυμα κρυπτογραφημένο σε RSA:

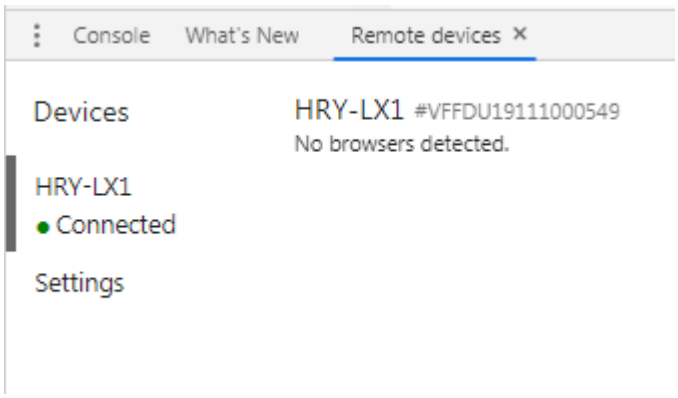


Εάν δεν εμφανιστεί το συγκεκριμένο παράθυρο τότε τα πράγματα για τη συγκεκριμένη συσκευή δεν είναι τόσο εύκολα. Σε κάποιες περιπτώσεις θα χρειαστεί αναζήτηση της διαδικασίας ρυθμίσεων στο διαδίκτυο. Ένα link ([γ], https://developers.google.com/web/tools/chrome-devtools/remote-debugging) είναι αυτό που προσφέρει η ίδια η Google και αναλύει την όλη διαδικασία επιγραμματικά καθώς και ένα άλλο([δ], https://stackoverflow.com/questions/21925992/chrome-devtools-devices-does-not-detect-device-when-plugged-in) από τον ιστότοπο του stackoverflow.com για σίγουρη λειτουργία.

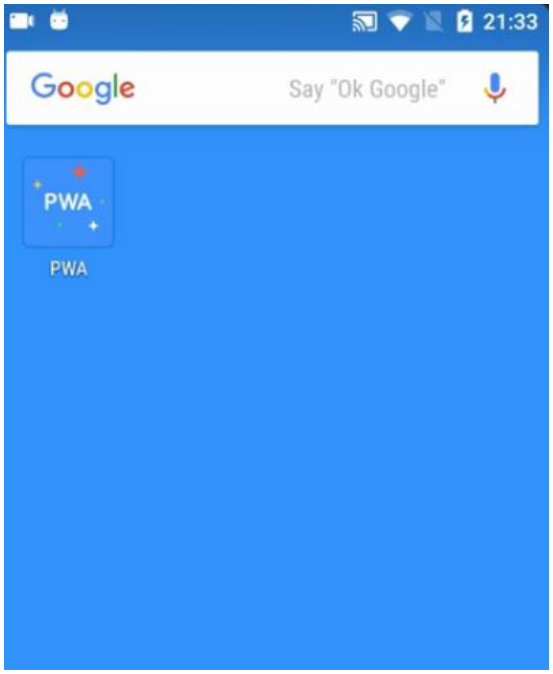


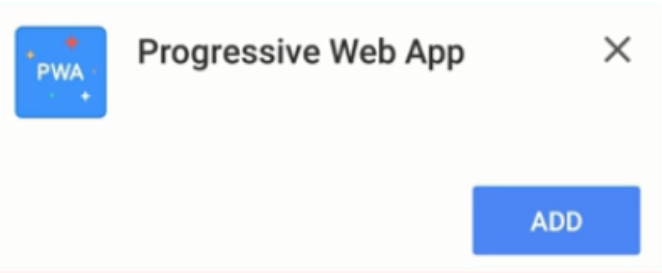
Χρειάζεται εγκατεστημένο στο desktop το λογισμικό της εκάστοτε συσκευής. Πίσω στον Chrome, πάνω δεξιά στα devtools επιλέγεται το control menu, επιλέγεται το ‘More tools’. Στις επιλογές που εμφανίζονται επιλέγεται το ‘Remote devices’. Εκείνο εμφανίζει ένα νέο παράθυρο που αφορά τις συνδεδεμένες συσκευές. Επιλέγοντας το πρώτο checkbox ‘Discover USB devices’ και αν η εφαρμογή δεν φιλοξενείται από κάποιον δικτυακό host, θα επιλεγεί και το δεύτερο checkbox ‘Port forwarding’ ώστε να κατευθυνθεί η εξωτερική συσκευή στην localhost port.

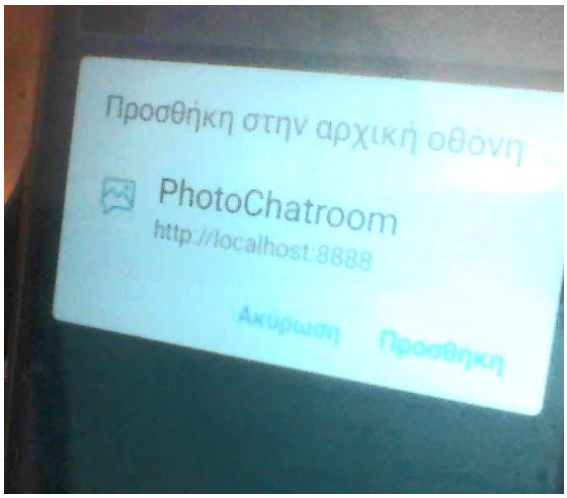
Αυτά ήταν τα βήματα για όσα χρειαζονται στα πλαίσια της διαμόρφωσης ώστε να λειτουργήσει σωστά το USB Debugging. Όταν η σύνδεση με την εξωτερική συσκευή γίνει επιτυχώς, στο Remote devices θα εμφανιστεί το όνομα της συσκευής και η κατάσταση της. Στη συσκευή μπορεί να εκτελεστεί η εφαρμογή από τον browser μέσα από το ίδιο URL.



Μόλις φορτωθεί η εφαρμογή στον browser της εξωτερικής συσκευής, παίρνει θέση το App Manifest και παρουσιάζει ένα notification για την εγκατάσταση της εφαρμογής στη συσκευή, τη μετατροπή της δηλαδή από web app σε native(-like) app, παρουσιάζοντας στο notification το όνομα της εφαρμογής και το icon που έγινε εισαγωγή στο App Manifest αρχείο.







Πατώντας το κουμπί ADD/Προσθήκη, εμφανίζεται η επιλογή να δωθεί ένα shortcut name στο app και στη συνέχεια λαμβάνεται ειδοποίηση πως το app εγκαταστάθηκε και είναι προσβάσιμο πλέον από το homescreen.

Τρέχοντας την εφαρμογή, παρατηρείται το splash screen το οποίο γίνεται generate απο το App Manifest, έχοντας το δηλωμένο background χρώμα, το όνομα της εφαρμογής και το icon να το απαρτίζουν.

8.3 Safari App Icons

Η Apple είναι η τελευταία εταιρεία που πρόσθεσε υποστήριξη για τα PWAs στο iOS λειτουργικό σύστημα. Ο safari browser και το iOS εξακολουθούν να μην είναι τόσο πλήρες όσο είναι οι browsers και τα λειτουργικά των άλλων μεγάλων εταιρειών. Τα δύο βασικά πεδία που υπάρχουν περιορισμοί είναι η χωρητικότητα της cache (εώς 50 Megabytes) και το App Manifest - ή πιο συγκεκριμένα, τα homescreen icons και τα launch images.

Ένα εφεδρικό σχέδιο για την προσθήκη των εικόνων σε iOS είναι μέσω του ιστοτόπου Human Interface Guidelines της Apple για τη δημιουργία εικονιδίων σε όλα τα διαφορετικά sizes και resolutions οθόνης μιας Apple συσκευής.

Εναλλακτικά, πρέπει να γίνει χρήση ενός οποιουδήποτε ιστοτόπου που παράγει εικονίδια, δηλαδή εικόνες μικρών διαστάσεων σε .ico μορφή. Το FAVIC-O-MATIC είναι ένας από αυτούς τους ιστοτόπους. Επιλέγοντας το ‘all sizes’ και κάνοντας upload το app icon, προχωράει σε auto-download και παρουσιάζει ένα generated HTML κώδικα για copy-paste στό head του HTML αρχείου της εφαρμογής.


```
<link rel="apple-touch-icon-precomposed" sizes="57x57" href="apple-touch-icon-57x57.png" />
<link rel="apple-touch-icon-precomposed" sizes="114x114" href="apple-touch-icon-114x114.png" />
<link rel="apple-touch-icon-precomposed" sizes="72x72" href="apple-touch-icon-72x72.png" />
<link rel="apple-touch-icon-precomposed" sizes="144x144" href="apple-touch-icon-144x144.png" />
<link rel="apple-touch-icon-precomposed" sizes="60x60" href="apple-touch-icon-60x60.png" />
<link rel="apple-touch-icon-precomposed" sizes="120x120" href="apple-touch-icon-120x120.png" />
<link rel="apple-touch-icon-precomposed" sizes="76x76" href="apple-touch-icon-76x76.png" />
<link rel="apple-touch-icon-precomposed" sizes="152x152" href="apple-touch-icon-152x152.png" />
```

Ωστόσο, πρέπει να γίνει εισαγωγή και για τις υπόλοιπες περιπτώσεις εκτός της iOS (τα εικονίδια για Android, Desktop κλπ). Παρακάτω εισάγονται τα αντίστοιχα favicon σε έναν “web-icons” φάκελο και καλούνται στο <head> μέτά τα “apple-touch-icon”.

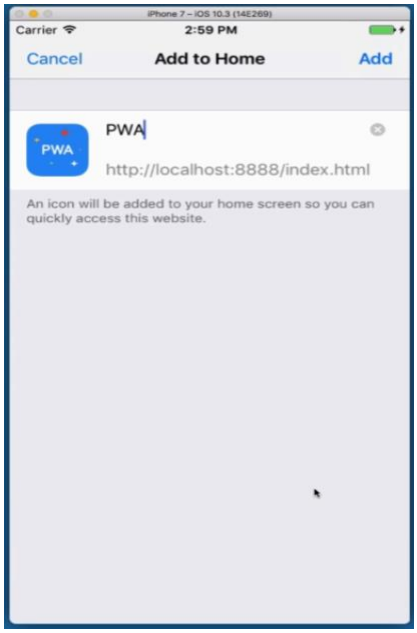
```
<link rel="icon" type="image/png" href="/web-icons/favicon-196x196.png" sizes="196x196" />
<link rel="icon" type="image/png" href="/web-icons/favicon-96x96.png" sizes="96x96" />
<link rel="icon" type="image/png" href="/web-icons/favicon-32x32.png" sizes="32x32" />
<link rel="icon" type="image/png" href="/web-icons/favicon-16x16.png" sizes="16x16" />
<link rel="icon" type="image/png" href="/web-icons/favicon-128.png" sizes="128x128" />
```

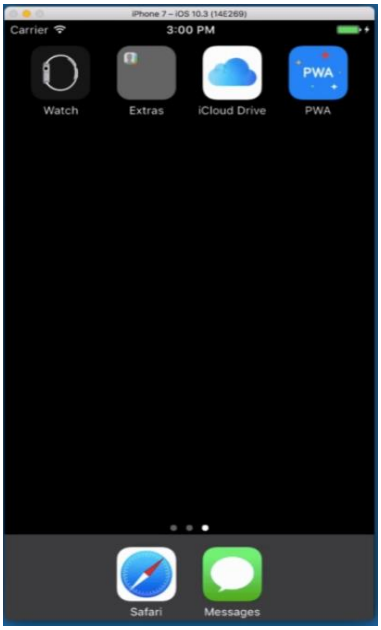
Για το Launch Screen, το εφεδρικό του γνωστού Splash Screen, προστίθεται ο παρακάτω κώδικας στο head του HTML αρχείου.

```
<!-- Apple launch image (splash screen) -->
<link rel="apple-touch-startup-image" href="/launch.png">

<!-- Apple launch title (app name) -->
<meta name="apple-mobile-web-app-title" content="PWA">

<!-- Hide Safari web UI -->
<meta name="apple-mobile-web-app-capable" content="yes">
```

Το app μπορεί να τρέξει μέσα από έναν iOS simulator. Προτείνεται να χρησιμοποιηθεί το Lambda Test app (https://www.lambdatest.com/) για να τρέξει ένα URL σε διαφορετικά browsers και

συσκευές. Σε έναν εικονικό safari, δίνοντας τη διεύθυνση http://localhost:8888, εμφανίζεται το notification για την εγκατάσταση της εφαρμογής στον προσομοιωτή, για παράδειγμα μιας iPhone 7 συσκευής. Σε αυτό το στάδιο ο χρήστης μπορεί να δώσει shortcut name που θα εμφανίζεται κάτω από το εικονίδιο του νέου εγκατεστημένου app.





Πατώντας πάνω δεξιά το OK και γυρνώντας στο homescreen, το βλέπουμε εγκατεστημένο στη συσκευή.

Όταν ο χρήστης ανατρέξει την εφαρμογή, το Launch Screen παρουσιάζεται για λίγα δευτερόλεπτα κι έπειτα κρύβεται εμφανίζοντας έτσι το homescreen της εφαρμογής.

ΚΕΦΑΛΑΙΟ 9

PhotoMessage Progressive Web App

Σε αυτό το κεφάλαιο, κατασκευάζεται μια chat εφαρμογή στο Web χρησιμοποιώντας όσες τεχνολογίες και τεχνικές αναφέρθηκαν καθώς θα καλυφθούν όλες οι προδιαγραφές που σχετίζονται με την **απόδοση** και την **εμπειρία χρήστη.** Μετά την ανάπτυξη της, η εφαρμογή θα καταταχθεί δικαίως στην κατηγορία των **PWAs**. Ένας χρήστης θα συμμετέχει σε ένα chat room και θα έχει την ικανότητα να τραβήξει φωτογραφία, θα την επισυνάπτει με ένα μήνυμα και θα την στέλνει στο κοινό chat room σε πραγματικό χρόνο. Παράλληλα, θα μπορεί να διαβάζει μηνύματα άλλων χρηστών που επισκέφτηκαν το chat room της εφαρμογής και που στη συνέχεια έστειλαν ένα μήνυμα με φωτογραφία. Το σύνολο των μηνυμάτων θα αποθηκεύεται στην εφαρμογή και θα φορτώνεται κατά την εισαγωγή του χρήστη στο chat room.

9.1 Επισκόπηση Τεχνολογιών

Σε αυτό το project χρησιμοποιείται το **Node.js** και η βιβλιοθήκη **Express** στο back-end, καθώς εκτός των άλλων, θα ανέβει ένας απλός Web Socket server για να κάνει την εφαρμογή να τρέχει real-time. Επίσης, για το front-end θα χρησιμοποιηθούν οι βιβλιοθήκες της Bootstrap και της JQuery για τον γρήγορο και απλό σχεδιασμό της εφαρμογής αφού γίνεται συγκέντρωση στο να προβληθεί ο σχεδιασμός που απαιτεί η PWA αρχιτεκτονική και χρειάζεται απλότητα στην ποικιλία των web υπηρεσιών που χρησιμοποιούνται. Στη συνέχεια, θα αναπτυχθεί ένας push server που θα διαχειρίζεται τα Push notifications. Μετά γίνεται εισαγωγή της authentication λειτουργίας στο back-end με τη βοήθεια του Passport.js, απαραίτητη για την ασφάλεια των προσωπικών δεδομένων του user καθώς επίσης και για την προστασία σημείων του ιστοτόπου που η πρόσβαση σε αυτά επιτρέπεται μόνο σε εξουσιοδοτημένους users. Τέλος, χρησιμοποιείται το Git

Bash για τερματικό και το yarn αντί του nmp ως resource-package manager του Node.js που θα κάνει εγκατάσταση πόρους απ΄το διαδίκτυο στην εφαρμογή.

9.2 Εγκατάσταση

Δημιουργείται ένας φάκελο που ονομάζεται PhotoMessageApplication.

Από τον ιστότοπο του yarn (https://yarnpkg.com) γίνεται download του installer για την πλατφόρμα που χρησιμοποιείται. Μετά την εγκατάσταση του στον υπολογιστή, στον terminal γίνεται κατεύθυνση στο φάκελο PhotoMessageApp που δημιουργήθηκε. Εκεί, τρέχει η εντολή yarn init και θα πρέπει να συμπληρωθούν τα πεδία που παρουσιάζει το yarn στον προγραμματιστή και που αφορούν την εφαρμογή.

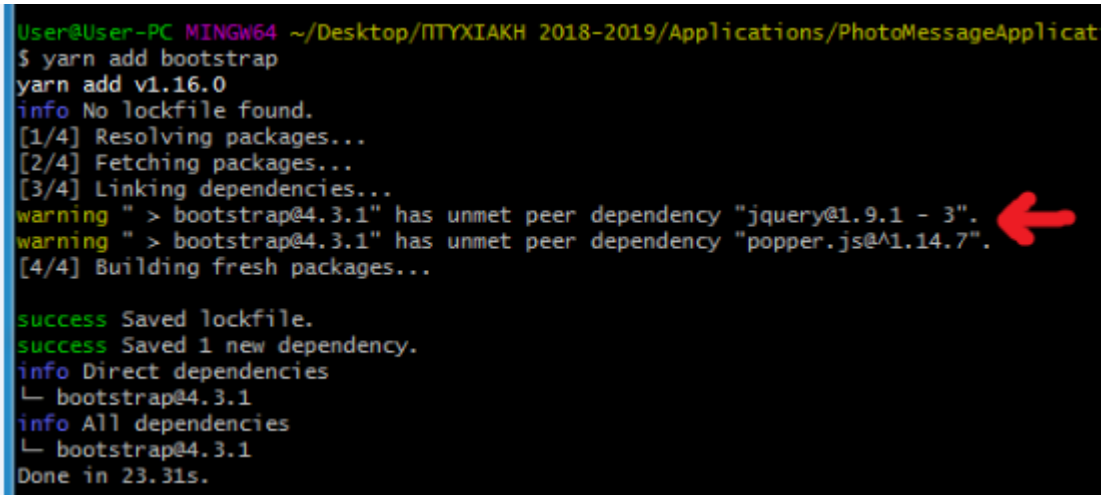
```
User@User-PC MINGW64 ~/Desktop/ΠΤΥΧΙΑΚΗ 2018-2019/Applications/PhotoMessageApplication
$ yarn init
yarn init v1.16.0
question name (PhotoMessageApplication): stef
question version (1.0.0): 1.0.0
question description: simple photo-caption PWA
question entry point (index.js): server/
question repository url:
question author: StefanosKokkalis
question license (MIT):
question private:
success Saved package.json
Done in 80.92s.

User@User-PC MINGW64 ~/Desktop/ΠΤΥΧΙΑΚΗ 2018-2019/Applications/PhotoMessageApplication
$
```

Αυτόματα, δημιουργείται ένα αρχείο package.json με τις πληροφορίες της εφαρμογής καθώς και πληροφορίες όπως τα “dependencies”, δηλαδή τα packages που απαιτούντε από την εφαρμογή και που αντιστοιχούνται με την έκδοση που χρειαζεται και που θα γίνονται αυτόματα εγκατάσταση όταν ένας προγραμματιστής τρέξει την εντολή `yarn add` / `npm install` στο φάκελο της εφαρμογής.

```json
{
  "name": "PhotoChatroom",
  "version": "1.0.0",
  "description": "simple photo-caption PWA",
  "main": "server/",
  "author": "StefanosKokkalis",
  "license": "MIT",
  "scripts": {
    "start": "node server/"
  },
  "dependencies": {
    "bootstrap": "^4.3.1",
    "express": "^4.17.0",
    "jquery": "^3.4.1",
  }
}
```

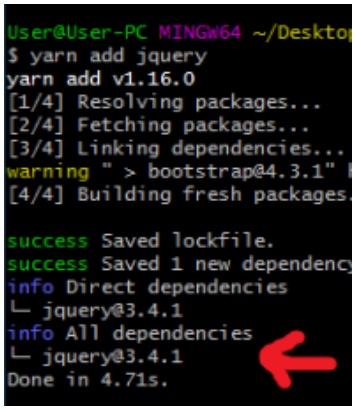
package.json

Στη συνέχεια, γίνονται εγκατάσταση οι βιβλιοθήκες των Bootstrap και JQuery. Τρέχοντας την εντολή `yarn add bootstrap`, το yarn προχωράει στην εγκατάσταση του Bootstrap και αυτόματα εγκαθίσταται και η JQuery.

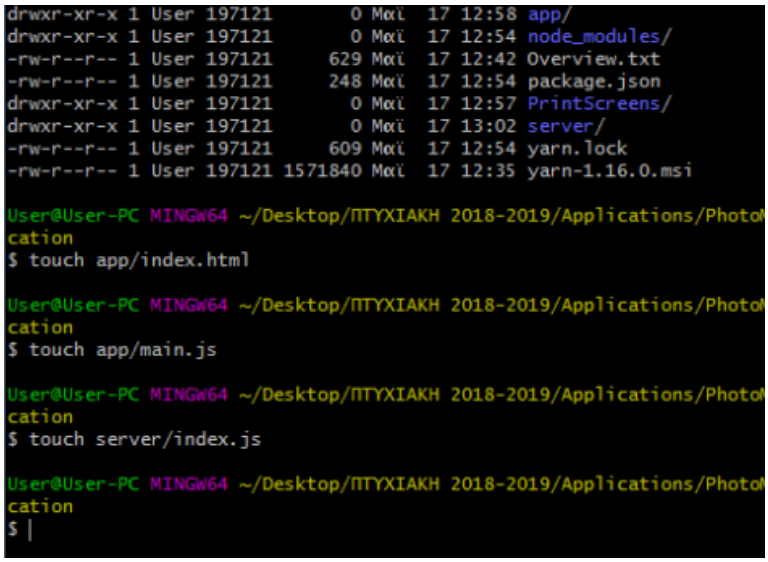


Παρόλο που η JQuery μπήκε αυτόματα, η έκδοση της είναι αρκετά παλιά. Γι αυτό θα τρέξει η εντολή `yarn add jquery`. Παρατηρείται πως το update έγινε με επιτυχία.

Στη συνέχεια, θα πρέπει να δημιουργηθεί ο φάκελος app που αφορά το front-end καθώς και ο φάκελος server. Με την εντολή `ls -l` παρατηρείται το περιεχόμενο του project και επιβεβαιώνεται την ύπαρξη των φακέλων που μόλις δημιουργήθηκαν.



Μέσα στον φάκελο app θα δημιουργηθεί το index.html και το room.js, τα βασικά αρχεία του client. Αντίστοιχα, στο φάκελο server το αρχείο index.js.



Για τον server θα γίνει εγκατάσταση το Express.js(https://expressjs.com), μία open-source βιβλιοθήκη του Node.js με την εντολή `yarn add express`. Το Node.js διαθέτει μια ενσωματωμένη λειτουργική μονάδα που ονομάζεται HTTP module, η οποία του επιτρέπει να μεταφέρει δεδομένα μέσω του πρωτοκόλλου HTTP (Hyper Text Transfer Protocol).

Για την εγκατάσταση του server της εφαρμογής, μέσα στο server/index.js αρχείο και γίνει η προσθήκη του παρακάτω κώδικα:

```
//Static Express Server

const express = require('express');

const http = require('http');


//Create HTTP Server

const app = express();

const server = http.Server(app);


//Server "app" directory

app.use(express.static(`${__dirname}/../app`));


//Server "node_modules" directory
```

```
app.use('/modules',

express.static(`${__dirname}/../node.modules'));


//Start Server

server.listen( 8888, () => console.log('Photo Message running on localhost:8888'));
```

Με τον τρόπο αυτό, δημιουργήθηκε ένας server που τρέχει όταν δώσουμε την `yarn start` εντολή στον terminal και στη συνέχεια εισάγουμε το URL `http://localhost:8888` στον browser. Ο root φάκελος του app προστατεύεται και δεν εκτίθεται καθώς και ο φάκελος node_modules θα γίνεται χρήση απο το app με το όνομα `modules`.

Για να κάνει auto-run ο server όταν τρέχει το app, προστίθεται στο **package.json** το “scripts” object και μέσα του μία μεταβλητή “start” με value το string “node server/”. Μόλις η εφαρμογή ξεκινήσει, ο Node.js, θα αναζητησει το index.js στο φάκελο `server/` και μόλις το βρει θα το εκτελέσει.

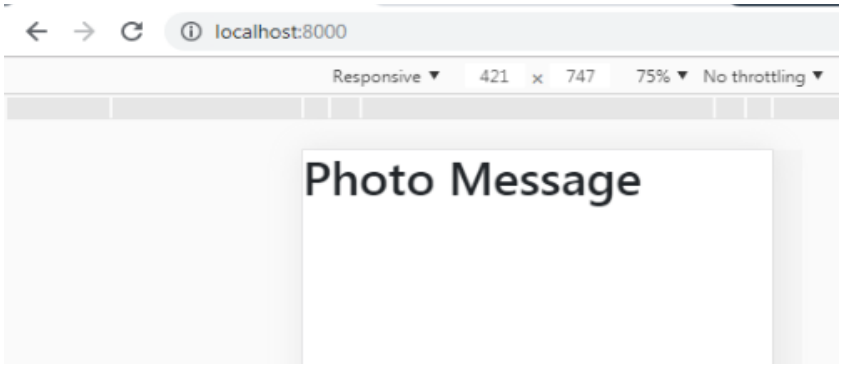
```
{
  "name": "stef",
  "version": "1.0.0",
  "description": "simple photo-caption PWA",
  "main": "server/",
  "author": "StefanosKokkalis",
  "license": "MIT",
  // added new scripts section
  "scripts": {
      "start": "node server/"
  },
  //////
  "dependencies": {
    "bootstrap": "^4.3.1",
    "express": "^4.17.0",
    "jquery": "^3.4.1"
  }
}
```

Στη συνέχεια πρέπει να χτιστεί η HTML και να μπει η επικεφαλίδα “Photo Message” μέσα στο body, καλώντας παράλληλα τα scripts που χρειάζονται από τον φάκελο modules, δηλαδή τον κρυμμένο φάκελο node_modules.

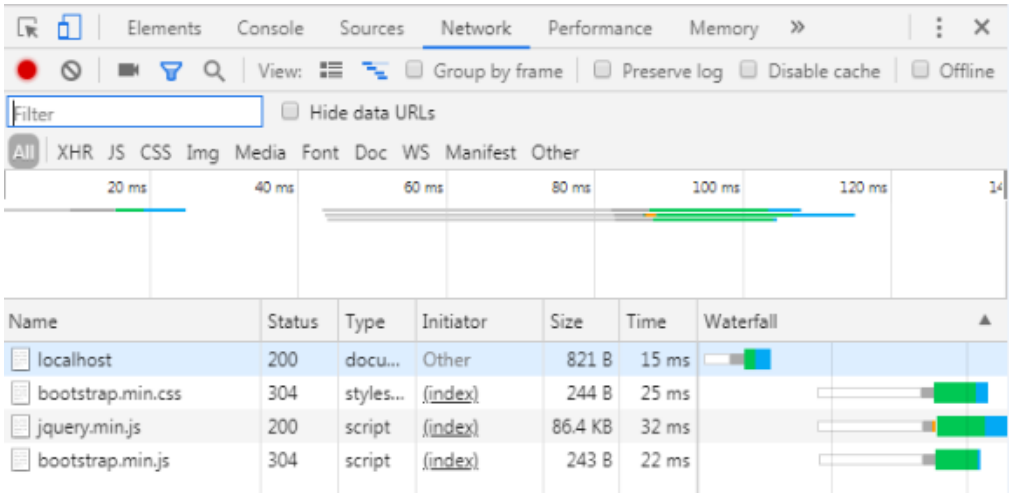
index.html

```
<html>
<head>
    <title>Photo Message</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="mobile-web-app-capable" content="yes">

    <!-- Styles -->
    <link rel="stylesheet" href="/modules/bootstrap/dist/css/bootstrap.min.css">

 </head>
<body>
    <h1>Photo Message</h1>

    <!-- Scripts -->
    <script src="/modules/jquery/dist/jquery.min.js"></script>
    <script src="/modules/bootstrap/dist/js/bootstrap.min.js"></script>

</body>
</html>
```

HTML μόνο με τίτλο και έναν header για content, καθώς καλεί και τις βιβλιοθήκες bootstrap & JQuery



Το backbone της εφαρμογής είναι έτοιμο. Πηγαινοντας στο Network tab των devtools θα παρατηρηθεί πως έχουν φορτωθεί οι πόροι που εώς τώρα καλούνται από την HTML.



9.3 Δημιουργώντας το ‘App Shell’

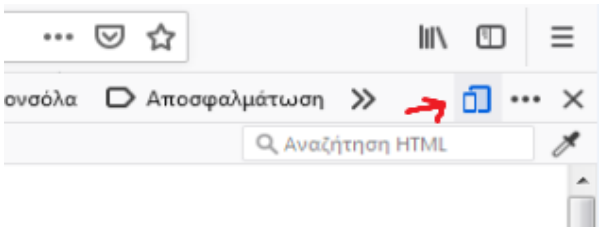
Παρακάτω ο κώδικας HTML που αφορά την στατική εμφάνιση και τη δομή της εφαρμογής ή αλλιώς το ‘APP SHELL’, το κέλυφος που αγκαλιάζει το δυναμικό περιεχόμενο. Χρησιμοποιεί κλάσεις της Bootstrap βιβλιοθήκης για το styling της εφαρμογής, καθώς χρησιμοποιεί και τα font-awesome εικονίδια που παρέχονται.

```
<html>
<head>
    <title>Photo Message</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="mobile-web-app-capable" content="yes">
    <link rel="stylesheet" href="/modules/bootstrap/dist/css/bootstrap.min.css">
</head>
<!-- App Shell -->

<body class="bg-info">

    <main class="p-2">

        <div id="messages" class="container-fluid">

            <div id="loader" class="text-center text-light" style="margin-top:10rem;">

                <i class="fas fa-spinner fa-2x fa-pulse"></i>

                <p>loading messages</p>

            </div>

        </div>

    </main>

    <div class="container-fluid fixed-bottom">

        <div id="toolbar" class="row bg-dark">

            <div class="col p-2 d-flex">

                <button id="camera" data-toggle="modal" data-target="#viewfinder" class="fas fa-camera fa-lg px-3 mr-2 text-light border-light bg-dark rounded-circle"></button>

                <textarea id="caption" placeholder="Image Caption..." maxlength="120" class="flex-grow-1 w-100 rounded border-light align-middle pr-5" style="resize:none;"></textarea>

                <a id="send" style="right:1.25rem;bottom:1.25rem; padding:0.5rem;" class="fas fa-paper-plane fa-lg position-absolute text-dark"></a>
```

```
                </div>
            </div>
        </div>
        <!-- Camera Modal -->
        <div id="viewfinder" class="modal fade" tabindex="-1" role="dialog" aria-hidden="true">
            <div class="modal-dialog fixed-bottom" role="document">
                <div class="modal-content bg-dark">
                    <div class="modal-body p-1 text-center">
                        <video id="player" autoplay height=”100” class="w-100 bg-white rounded
"></video>
                        <button data-dismiss="modal" type="button" id="shutter" class="fas fa-circle
fa-3x rounded-circle bg-white text-danger p-1 my-2"></button>
                    </div>
                </div>
            </div>
        </div>
        <!-- Photo Modal -->
        <div class="modal fade" id="photoframe" tabindex="-1" role="dialog" aria-hidden="true">
            <div class="modal-dialog" role="document">
                <div class="modal-content">
                    <div class="modal-body p-1">
                        <img src="" class="w-100 rounded" data-dismiss="modal">
                    </div>
                </div>
            </div>
        </div>
<script src="/modules/jquery/dist/jquery.min.js"></script>
```

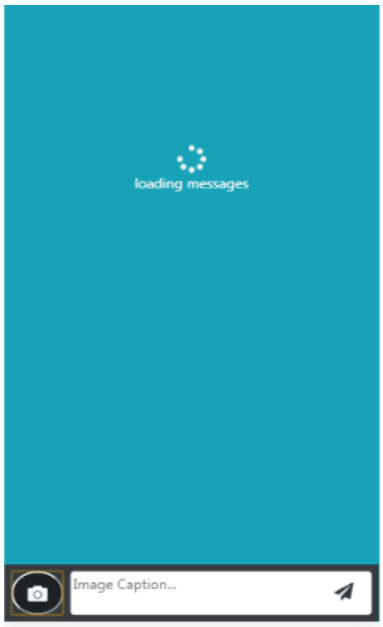
```
<script src="/modules/bootstrap/dist/js/bootstrap.min.js"></script>
</body>
<!-- App Shell -->
</html>
```

Στο view της εφαρμογής από τον browser, έχοντας ανοιχτά τα devtools στα δεξιά, επιλέγεται το view app as mobile device.

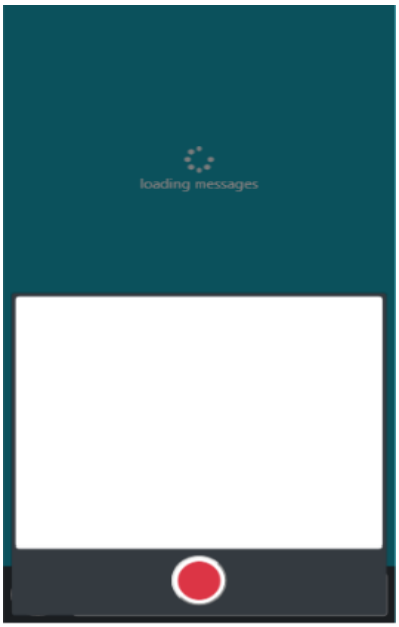


Το App Shell αποτελείται από ένα στατικό container για την εμφάνιση των δυναμικών μηνυμάτων και άλλο ένα στατικό και fixed div στο bottom της εφαρμογής το οποίο περιέχει ένα button για την κάμερα, ένα textbox για το μήνυμα που θα συνοδεύσει τη φωτογραφία και τέλος ένα button που στέλνει το μήνυμα.

Επίσης, αποτελείται και από δύο modal που αφορούν την διαδικασία που χρειάζεται ο χρήστης να κάνει capture με την κάμερα και την εμφάνιση των φωτογραφιών των μηνυμάτων που θα βρίσκονται στο chat room.



App Shell



App Shell Modal

9.4 Προσθήκη Λειτουργικότητας

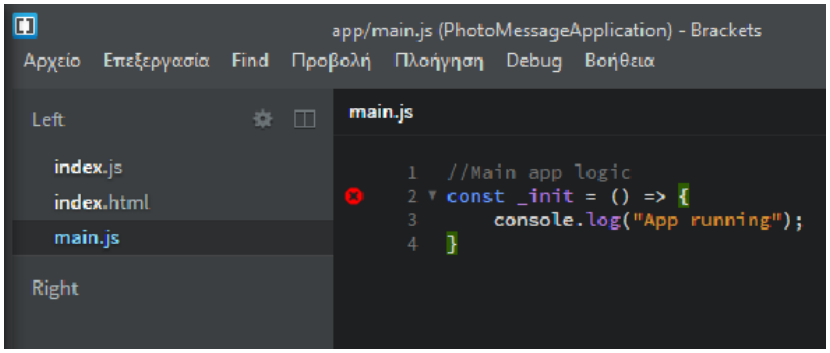
Το επόμενο βήμα θα είναι να προστεθεί στο φάκελο /app το αρχείο room.js αλλά πρώτα θα πρέπει να γίνει ένας έλεγχος υποστήριξης του media support API που παρέχει η HTML5. Εάν δεν υποστηρίζεται, θα εμφανίζεται αντίστοιχο μήνυμα στην εφαρμογή. Σε mobile δυστυχώς, απαιτείται το πρωτόκολλο `https://` για να υποστηριχθεί το media support API ολοκληρωμένα αλλά μέσα από το USB debugging και τα remote devices του Chrome που αναφέρθηκαν στο προηγούμενο κεφάλαιο, η εφαρμογή θα τρέξει κανονικά στο localhost.

Στο τέλος του body και κάτω από τα scripts, γίνεται προσθήκη ενός <script> που θα περιέχει τον κώδικα του ελέγχου συμβατότητας και που θα καλεί την _init() μέσα από το room.js εάν τελικά το mediaDevices είναι συμβατό:

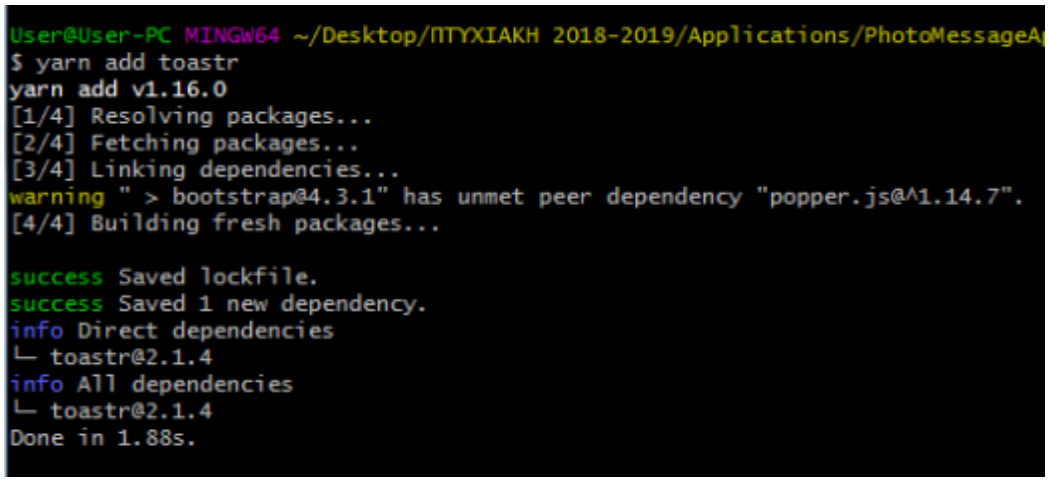
```
<!-- Init App if Supported -->
<script>
// Check Compatibility
if( 'mediaDevices' in navigator ){
        //call init() from room.js
        _init();
} else {  console.log("App not supported.");   }
</script>
```

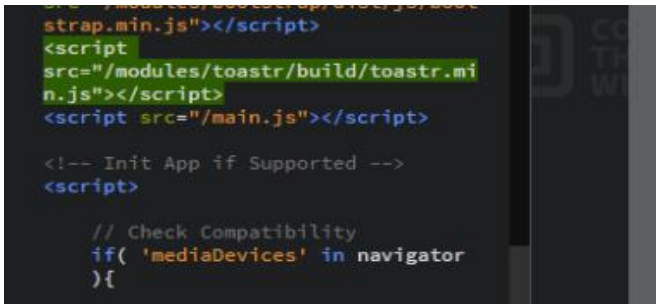
Στο room.js κατασκευάζεται η μέθοδος init().

```
//Main app logic
const _init = () => {
   console.log(“App running”);
}
```



Για τις ειδοποιήσεις στον user θα χρησιμοποιείται το toastr που προσφέρει εύκολη εισαγωγή και όμορφο design. Γίνεται download και εγκατάσταση εκτελώντας την εντολή `yarn add toastr` από τον terminal. Στη συνέχεια προστίθεται η προέλευση του script στην HTML.





Στο head της HTML γίνεται προσθήκη:

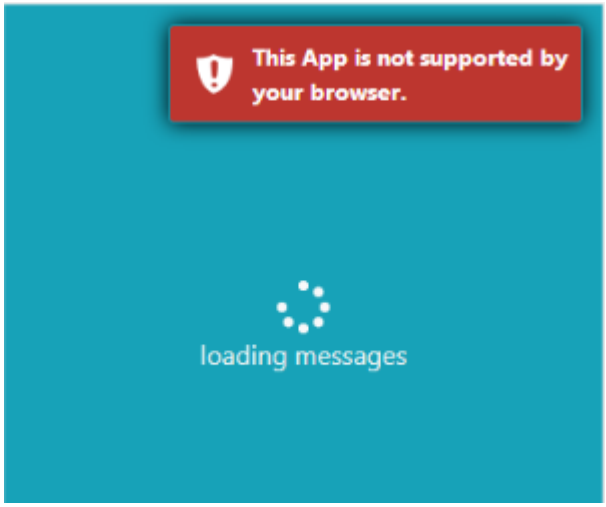
```
<style>
    #toast-container {
        top: 0;
        left: 0;
        width: 100%;
    }
    #toast-container > div {
        opacity: 1;
        box-shadow: 0 0 2rem rgba(0,0,0,0.75);
        border-radius: 0;
```

```
        width: 100%;
        max-width: 100%;
    }
</style>
```

Για την πρώτη ειδοποίηση και το error handling, θα αναγκαστεί να γίνει false το if statement για το compatibility *βάζοντας mediaDevices_fake αντι mediaDevices*, με τον εξής τρόπο:

```
//Check compatibility
if( 'mediaDevices_fake' in navigator ){
    //call init() from room.js
    _init();
} else {
    console.log("error");
    //Show notification
    toastr.error( null, "This App is not supported by your browser.", {timeout: 3000});
}
```

Τρέχοντας το app, θα εμφανιστεί στον browser η ειδοποίηση που δώθηκε στο toastr.error(null, "") όπως φαίνεται και από την εικόνα στα δεξιά.



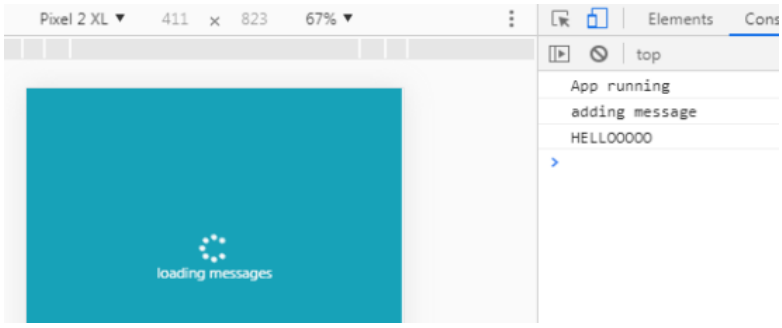
Το toastr λειτουργεί. Στην HTML θα γυρίσει πίσω το mediaDevices και στο room.js προσθέτοντας τον παρακάτω κώδικα μέσα στην _init():

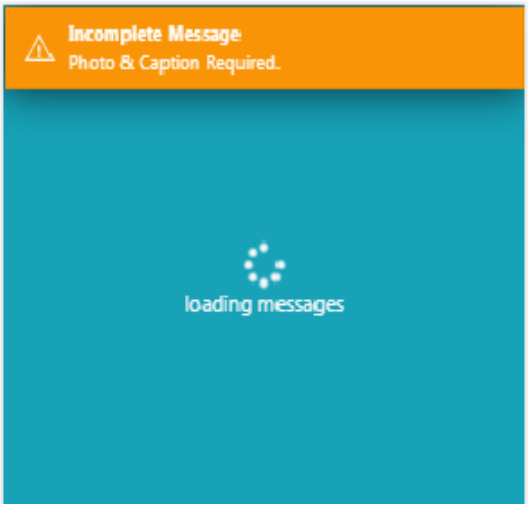
```
//Switch on camera in viewfinder
    $("#viewfinder").on("show.bs.modal", () => {
        console.log("camera on");
    });
```

```
    $("#viewfinder").on("hidden.bs.modal", () => {
        console.log("camera off");
    });

    //Take Photo
    $("#shutter").on("click", () => {
        console.log("take photo");
    });

    //Submit Photo
    $("#send").on("click", () => {

        //Get caption text
        let caption = $("#caption").val();

        //Check message if ok
        if( !caption) {
            //Show notification and return
            toastr.warning('Photo & Caption Required.', 'Incomplete
Message');
            return;
        }
        console.log(‘adding message’);
        console.log(caption);
```

```
});
```

Βάζοντας κείμενο στο textbox και πατώντας το send παρατηρείται στην κονσόλα:





9.5 Κάμερα

Για την λειτουργία της κάμερας και την πρόσβαση της από την συσκευή του client, κατασκευάζεται μία JavaScript κλάση που λειτουργεί σαν μία αυτόνομη ύπαρξη που όποτε καλείται στον κώδικα δημιουργεί ένα στιγμιότυπο με όλα τα χαρακτηριστικά που ανήκουν στην κλάση αυτή. Δημιουργείται ένας νέος φάκελος “classes” μέσα στον φάκελο app. Μέσα στον νέο φάκελο “classes” δημιουργείται το αρχείο “camera.js”.

```
// Camera’s Class

class Camera {

    constructor( video_node ){

        //Camera stream DOM node

        this.video_node = video_node;

        console.log(‘camera instance created’);

    }

}
```

Για να ενεργοποιηθεί η κλάση της κάμερας, καλείται στο <head> του αρχείου HTML.

<script src="/classes/camera.js"></script>

Απαιτείται η σύνδεση της κλάσης με το video στοιχείο που βρίσκεται στο camera modal και που ορίστηκε μέσω του id κλειδιού ‘player’ το οποίο μπορεί επιλεχθεί εύκολα με την βοήθεια των selectors της JQuery. Στο room.js θα προσθέσω πάνω από την _init() :

```
// Init new camera instance on the video player
const camera = new Camera( $("#player")[0] );
```

Ανανεώνονται τα switch on και switch off στο room.js:

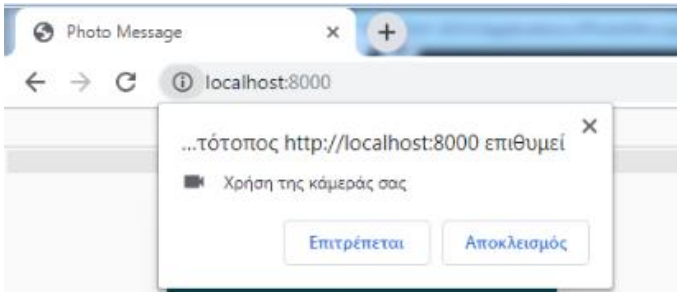
```
//Switch on camera in viewfinder
    $("#viewfinder").on("show.bs.modal", () => {
        //console.log("camera on");
        camera.switch_on();
    });

    $("#viewfinder").on("hidden.bs.modal", () => {
        //console.log("camera off");
        camera.switch_off();
    });
```

Πίσω στο camera.js αρχείο, κατασκευάζεται η λειτουργία switch_on(). Με την getUserMedia() χειραγωγούνται το βίντεο και ο ήχος. Δεδομένου πως το promise που επιστρέφεταιι γίνεται resolved απαιτείται η τροφοδοσία του video από την κάμερα η παρουσίαση του μέσα στο modal με id ‘viewfinder’ όποτε εκείνο είναι εμφανές στον user. Αντίστοιχα για τη λειτουργία switch_off(). Χρησιμοποιείται το stream σαν callback και καλείται μέσα από την switch_off(), μέσα από την getTracks() που επιστρέφει ένα array από τα media devices που χρησιμοποιούνται. Μόνο ένα στοιχείο του array που επιστρέφεται απαιτείται, το device που αφορά την κάμερα. Από την HTML αφαιρείται το height και η bg-white κλάση του video στοιχείου. Μέσα στο αρχείο camera.js και κάτω από τον constructor προστίθεται:

```
//camera feed viewfinder on
    switch_on() {
        //get camera media stream and set on <video>
```

```
        navigator.mediaDevices.getUserMedia({
            video: {width: 600, height: 600},
            audio: false
        }).then( stream => {
            this.video_node.srcObject = this.stream = stream;
        })
    }

    //camera feed viewfinder off
    switch_off() {
        // pause video node
        this.video_node.pause();
        //stop media stream
        this.stream.getTracks()[0].stop();
    }
```

Τρέχοντας την εφαρμογή, ο browser ζητάει άδεια από τον user για την χρήση της κάμερας.



Πατώντας το κουμπί “επιτρέπεται” και τεστάροντας την κάμερα παρατηρείται πως λειτουργεί κανονικά και προσφέρει ροή βίντεο στο modal.

Επόμενο βήμα είναι η εισαγωγή της λογικής του shutter button. Εκείνο, όποτε πατηθεί με ένα κλικ του user, προκαλεί το capture ένα στιγμιότυπου του video stream που προσφέρεται από την κάμερα και το μετατρέπει σε εικόνα, αποθηκεύοντας την σε ένα canvas στοιχείο που δημιουργείται εκείνη τη στιγμή στην HTML.

Κάτω από την switch_off() λειτουργία, προστίθεται:

```
 //camera photo from camera stream

    take_photo() {


        //create a <canvas> element to render the photo

        let canvas = document.createElement('canvas');

        canvas.setAttribute('height', 600);

        canvas.setAttribute('width', 600);


        // get camera context

        let context = canvas.getContext('2d');


        //draw (render) the image on the canvas

        context.drawImage(this.video_node, 0, 0, canvas.width, canvas.height);


        //get the canvas image as data url

        this.photo = context.canvas.toDataURL();


        //destroy canvas

        context = null;

        canvas = null;


        return this.photo;

    }
```

Στο room.js προστίθεται η λειτουργία camera.take_photo() που ενεργοποιείται όταν ο χρήστης κάνει κλικ στο shutter button μέσα στο modal. Επίσης μέσα στην ίδια λειτουργία, η photo μπαίνει ως background-image στο button της κάμερας, κάτω αριστερά στο bottom-fixed πλαίσιο της εφαρμογής, διότι είναι απαραίτητη η προεπισκόπηση της εικόνας από τον user πριν εκείνος την δημοσιεύσει στο chat room.

```
//Take Photo
```

```
    $("#shutter").on("click", () => {
        //console.log("take photo") ;
        let photo = camera.take_photo();
        //show photo preview in camera button
        $("#camera").css("background-image",
`url(${photo})`).addClass('withphoto');
    });
```

Αυτό που μένει, είναι η ανανέωση της λειτουργίας send() ώστε να γίνεται έλεγχος εάν η φωτογραφία υπάρχει κατί την δημοσιεύση της, καθώς να αφαιρείται το background-image του camera button και του κειμένου στο textbox.

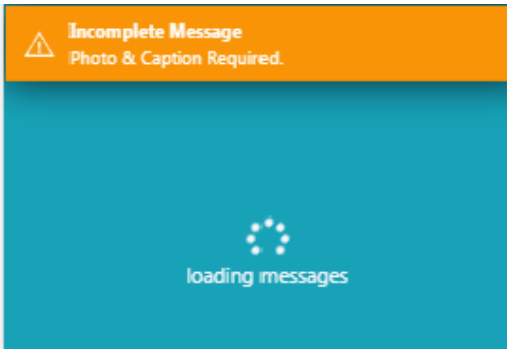
```
//Submit Photo
    $("#send").on("click", () => {
        //Get caption text
        let caption = $("#caption").val();
        //Check message if ok
        if(!camera.photo || !caption) {
```

```
            //Show notification and return
            toastr.warning('Photo & Caption Required.', 'Incomplete Message');
            return;
        }

         console.log('adding message');
         console.log(caption);

        // Reset caption field on success
         $('#caption').val('');
            $('#camera').css('background-image','').removeClass('withphoto');
        camera.photo = null;
    });
```

Προστίθεται στην HTML μέσα στο style:

```
#camera.withphoto{
    color: transparent !important;
    background-size: cover;
}
```

Πατώντας δεξιά το send, το textbox αδειάζει και η φωτογραφία φεύγει από το αριστερο shutter button.



Επίσης, κατά το send χωρίς κείμενο ή χωρίς φωτογραφία το error handling λειτουργει, παρουσιάζοντας στον χρήστη ένα warning toastr απαιτώντας την από κοινού παρουσία της φωτογραφίας και του κειμένου.

9.6 Κατασκευάζοντας τα μηνύματα

Όταν αποσταλεί από τον user μήνυμα που συνοδεύει φωτογραφία, το νέο μήνυμα που δημιουργείται εμφανίζεται στο #messages container. Στο room.js, μέσα στο send, κάτω από τα δύο console.log()

```
console.log('adding message');

console.log(caption);
```

,καλείται η renderMessage(message):

```
// add new message

let message = messages.add(camera.photo, caption);

console.log(messages.all);

// Render new message in feed

renderMessage({photo: camera.photo, caption: caption});

//renderMessage( message );
```

Για να χτιστεί η renderMessage(), έξω και κάτω από την send προστίθεται:

```
// Create new message element

const renderMessage = ({photo: camera.photo, caption: caption}) => {

    // Message HTML

    let msgHtml = `<div style="display:none;" class="row message bg-light mb-2 rounded
shadow">

    <div class="col-2 p-1">

        <img src="${ camera.photo }" class="photo w-100 rounded">

    </div>

    <div class="col-10 p-1">

      ${caption}

    </div></div>`;

    $(msgHtml).prependTo('#messages').show(500);

}
```

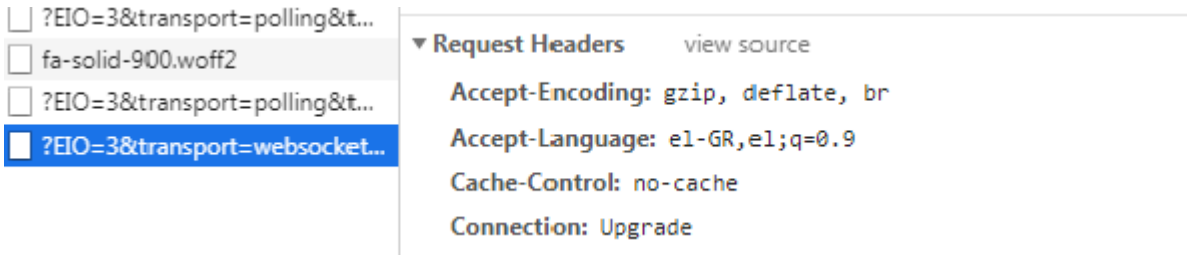
Όταν γίνεται κλικ από τον χρήστη στο message που υπάρχει στο container του chat room, εμφανίζεται το photo modal με τη φωτογραφία που ανήκει στο επιλεγμένο message. Κάτω από το renderMessage(message) προστίθεται:

```
// Show message photo in modal

const showPhoto = (e) => {

    console.log("Showing Photo");


    // get photo source

    let photoSrc = $(e.currentTarget).attr('src');


    // set to and show photo()

    $('#photoframe img').attr('src', photoSrc);

    $('#photoframe').modal('show');

}
```

Photo Modal opens on message click

9.7 Συγχρονίζοντας τα μηνύματα μέσω του socket.io

Για την ανάπτυξη μιας εφαρμογής συνομιλιών, απαιτείται η δημιουργία ενός συστήματος πραγματικού χρόνου για την αποστολή και λήψη δεδομένων. Με τις βασικές τεχνολογίες του web είναι αδύνατη η κατόρθωση της ζητούμενης κατασκευής. Χρησιμοποιείται η βιβλιοθήκη Socket.IO για αυτόν τον σκοπό. Εκείνο χρησιμοποιεί την τεχνολογία WebSocket, ένα διαδικτυακό πρωτόκολλο που παρέχει κανάλια πλήρους αμφίδρομης επικοινωνίας μέσω μιας TCP σύνδεσης. Το socket.io παρατηρεί ποιοί clients είναι συνδεδεμένοι στην εφαρμογή και επιτρέπει την κατασκευή λειτουργιών παραλληλίας. Για να εγκατασταθεί στην εφαρμογή εκτελείται στον terminal η εντολή `yarn add socket.io`.
Αφού γίνει η εγκατάσταση και πραγματοποιηθεί η κλήση του αρχείου στην HTML μέσα από την γραμμή <script src="/socket.io/socket.io.js" ></script>, θα επιβεβαιωθεί πως φορτώθηκε μέσα από το network tab των devtools.



Ανοίγοντας ένα ακόμη παράθυρο του browser και τρέχοντας την εφαρμογή από τον localhost:8888, στον terminal που λειτουργεί ως η κονσόλα του server, εμφανίζεται δεύτερο μήνυμα **'New Client Connected'** δηλώνοντας πώς αναγνωρίστηκε η προσέλευση μέσα από το δεύτερο παράθυρο.

Για τον συγχρονισμό των μηνυμάτων θα χρειαστεί μία νέα κλάση "Message" ώστε να χειρίζεται τα στιγμιότυπα της και τις λειτουργίες που απαιτούνται. Το object του message στο room.js είναι της ίδιας δομής που έχει η παράμετρος που απαιτεί η renderMessage({photo: camera.photo, caption}) λειτουργία και που είναι δηλωμένη στο room.js. Στο room.js θα γίνει εισαγωγή μέσα στην _init() και πάνω από το //Notify user of connection error:

```
// Init new message instance
    const messages = new Message();
```

Στη συνέχεια, θα δημιουργηθεί το αρχείο **message.js** μέσα στο φάκελο classes, και γίνεται προσθήκη το κομμάτι κώδικα:

```
class Message {
    constructor() {
        this.messages = [];
```

```
        //connect to socket server
        this.socket = io();
    }
    //Get all messages
    get all() {
        return this.messages;
    }
    //Add a new message
    add(data_url, caption_text) {
        //create message obj
        let message = {
            photo : data_url,
            caption : caption_text
        }
        //return formatted message obj
        return message;
    }
}
```

Στο renderMessage αλλάζει η παράμετρος από {photo: camera.photo, caption} σε message όπου message είναι τα δεδομένα που επέστρεψαν από την add(camera.photo, caption) που καλείται μέσα από το στιγμιότυπο messages. Στο room.js, ανανεώνεται το //Submit message σε:

```
    //Submit Photo
    $("#send").on("click", () => {
        //Get caption text
        let caption = $("#caption").val();

        //Check message if ok
        if(!camera.photo || !caption) {
            //Show notification and return
            toastr.warning('Photo & Caption Required.', 'Incomplete Message');
            return;
        }
```

```
        // add new message
        let message = messages.add (camera.photo, caption);
        console.log(messages.all);

        // Render new message in feed
        //renderMessage({photo: camera.photo, caption: caption});
        renderMessage( message );

        // Reset caption on success
        $('#caption').val('');
        $('#camera').css('background-image','').removeClass('withphoto');
        camera.photo = null;
    });
}
```

Επίσης, απαιτείται και η διόρθωση της κατασκευασμένης λειτουργίας. Έξω και κάτω από την send ανανεώνεται η renderMessage() σε:

```
// Create new message element
//const renderMessage = ({photo: camera.photo, caption: caption}) => {
const renderMessage = (message)=> {
    // Message HTML
    let msgHtml = `<div style="display:none;" class="row message bg-light mb-2 rounded
shadow">
    <div class="col-2 p-1">
        <img src="${ message.photo }" class="photo w-100 rounded">
    </div>
    <div class="col-10 p-1">
      ${message.caption}
    </div></div>`;
    $(msgHtml).prependTo('#messages').show(500);
}
```

Καλείται η νέα Message.js κλάση στην HTML.

Έπειτα στον server, χτίζεται ο web socket διακομιστής.

```
//Create web socket server

const io = socketio(server);

// Listen for new socket client (connection)

io.on(‘connection’, (socket) => {

    console.log(“New Client Connected”);

});
```

Για τη διαχείρηση των real-time δυνατοτήτων που παρέχει το socket.io απαιτείται η σύνδεση με τον socket server. Μέσα στο Message.js ανανεώνεται ο constructor ώστε όποτε καλεί τον socket server προσπαθεί να συνδεθεί με λάθος διακομιστή, καθώς κατασκευάζεται ένα event που αφορά το error handling σε περίπτωση που ο client δεν συνδεθεί επιτυχώς.

```
 constructor() {

        this.messages = [];

        //connect to socket server

        this.socket = io(“http://fakebook.com”);


        //Handle connection error

        this.socket.once('connect_error', () => {

            // Notify room.js via an event

            window.dispatchEvent( new Event('messages_error') );

        });

}
```

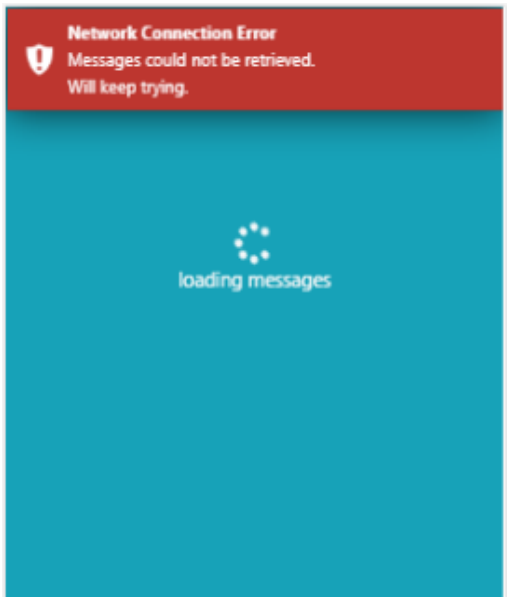
Στο room.js θα γίνει χειρισμός για το τι πρόκειται να γίνει στην περίπτωση που εμφανιστεί το ‘messages_error’ event. Μέσα στην _init() και κάτω από την δήλωση της μεταβλητής messages, προστίθεται:

```
 //Notify user of connection errors

  window.addEventListener('messages_error', () => {

        toastr.error('Messages could not be retrieved.<br>Will keep trying.', 'Network
Connection Error');

  });
```

Διορθώνεται το call του socket server σε:

```
//connect to socket server

this.socket = io();
```

Έπειτα, θα χρειαστεί να κατασκευαστεί ένα event για τη στιγμή που ένα νέο message δημιουργείται. Παράλληλα πρέπει η εφαρμογή να παρατηρεί εάν πυροδοτήθηκε αυτό το event και εάν το αντιληφθεί, η εφαρμογή να ενημερώνει τον socket server και εκείνος να αντιδράει με τη σειρά του αναλόγως.



Στο Message.js κάτω από το this.socket.once():

```
//listen for new message from server
 this.socket.on('new_message', (message) => {
      //add to local messages
      this.messages.unshift(message);


      //notify client via custom event
      window.dispatchEvent( new CustomEvent('new_message', {detail: message}) );
 });
```

Μέσα στην add()και κάτω από την δήλωση του message object, πάνω από το return προστίθεται:

```
// add to local messages
  this.messages.unshift(message);


  //emit to server
  console.log('sending to server');
  this.socket.emit('new_message', message);
```

Στο room.js προστίθεται μέσα στην _init() και κάτω από το //Notify user of connection errors:

```
//Listen for new message event
    window.addEventListener('new_message', (e) => {
        renderMessage( e.detail );
      });
```

Ο socket server όποτε παρατηρεί το event, ενεργοποιείται από την ενημέρωση που στέλνει ο client και μεταδίδει το message σε όλους τους clients με την προσφερόμενη από εκείνον λειτουργία broadcast(). Στο server/index.js ανανεώνεται το κομμάτι κώδικα του io.on(“connection”) σε:

```
// Listen for new socket client (connection)
io.on('connection', (socket) => {
    console.log("New Client Connected");
    //Send all messages to connecting client
    socket.emit('all_messages', messages);

    //listen for new messages
    socket.on('new_message', (message) => {

        //add to messages
        messages.unshift(message);

        //broadcast new message to all connected clients
        socket.broadcast.emit('new_message', message);
    });
})
```

Όταν η εφαρμογή ανοίξει σε δύο παράθυρα του Chrome, κάθε message που αποστέλλεται από τον πρώτο client, θα εμφανιστεί και στον δεύτερο καθώς αυτό ισχύει και αντιστρόφως.

Η εφαρμογή μετατρέπεται σε real-time αλλά εώς τώρα τα μηνύματα αποθηκεύονται προγραμματιστικά σε ένα array στον browser για όσο κρατάει το session του client. Αν ένας από τους δυο clients κάνει reload την εφαρμογή τα messages χάνονται. Δημιουργείται ένα JSON αρχείο για την αποθήκευση των messages. Όποτε ο user κάνει login στην εφαρμογή και εισέρχεται στο chat room, φορτώνονται όλα τα messages μέσα από το json. Το αρχείο κατασκευάζεται στον φάκελο server και ονομάζεται “chat_history.json”. Μέσα στο server/index.js γίνεται προσθήκη κάτω από το //Create web socket server:

```
// Init server messages from disk
const messageData = fs.readFileSync(`${__dirname}/chat_history.json`).toString();
const messages = messageData ? JSON.parse(messageData) : [];
```

Όταν ο server ξεκινήσει, διαβάζεται από τον express server το αρχείο “chat_history.json” με την fs.readFileSync() μέθοδο και μετατρέπεται σε string τύπο μέσα από την μέθοδο toString(),

αποθηκεύοντας τα δεδομένα στη μεταβλητή messageData. Στη συνέχεια, εάν είναι άδειο το JSON αρχείο δημιουργείται ένας άδειος πίνακας και αποθηκεύεται στη μεταβλητή “messages”, αλλιώς αποθήκευονται σε JSON τύπο τα messageData στη μεταβλητή messages.

Έπειτα, πάλι μέσα στο server/index.js, στον socket server και μέσα στο //listen for new messages, κάτω από το //add to messages, προστίθεται:

```
//Persist to disk

        fs.writeFileSync(`${__dirname}/chat_history.json`, JSON.stringify(messages));
```

Αφού έγινε εισαγωγή η ικανότητα να αποθηκεύονται τα messages σε ένα τοπικό αρχείο είναι χρήσιμο μόλις φορτώνεται η εφαρμογή να φορτώνονται από το αρχείο αυτό στον client όλα τα messages. Στο Message.js προστίθεται πάνω από την this.socket.on(‘new_message’), η λειτουργία this.socket.on(‘all_messages’) ώστε να λάβει η εφαρμογή όλα τα messages όταν συνδεθεί με τον server:

```
   //listen for all server messages (send or connect)

        this.socket.on('all_messages', (messages) => {


            //update local messages array

            this.messages = messages;


            //notify client via an event

            window.dispatchEvent( new Event('messages_ready') );

        });
```

Στο room.js, γίνεται εισαγωγή πάνω από τον window.addEventListener(‘new_message’) του αντίστοιχου listener για το ‘messages_ready’ event και θα εμφανιστούν τα messages στο στοιχείο με id “#messages” της HTML εάν υπάρχουν, με αντίθετη σειρά έχοντας τα νεότερα messages στοιχισμένα προς τα πάνω.

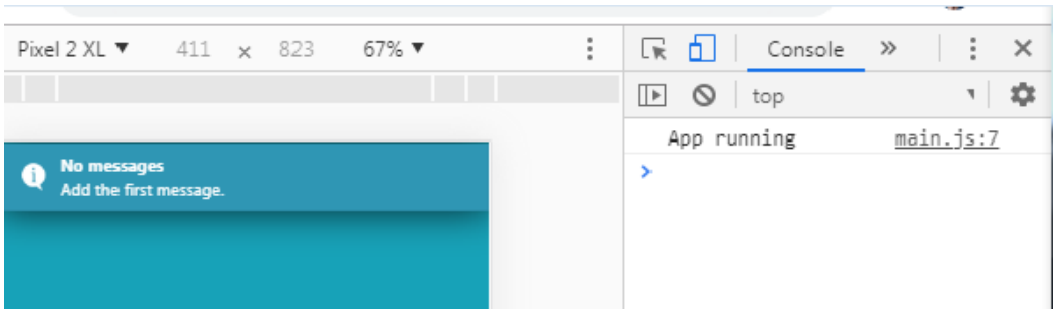
```
//Listen for existing messages from server

    window.addEventListener('messages_ready', () => {

        //remove the loader

        $('#loader').remove();


        //check if messages exist

        if ( messages.all.length == 0) {

           toastr.info('Add the first message.', 'No messages');
```

```
        }
        //Empty our existing messages if this update is from a reconnection
        $('messages').empty();

        //loop and render all messages [reverse as were prepending]
        messages.all.reverse().forEach( renderMessage );
    });
```

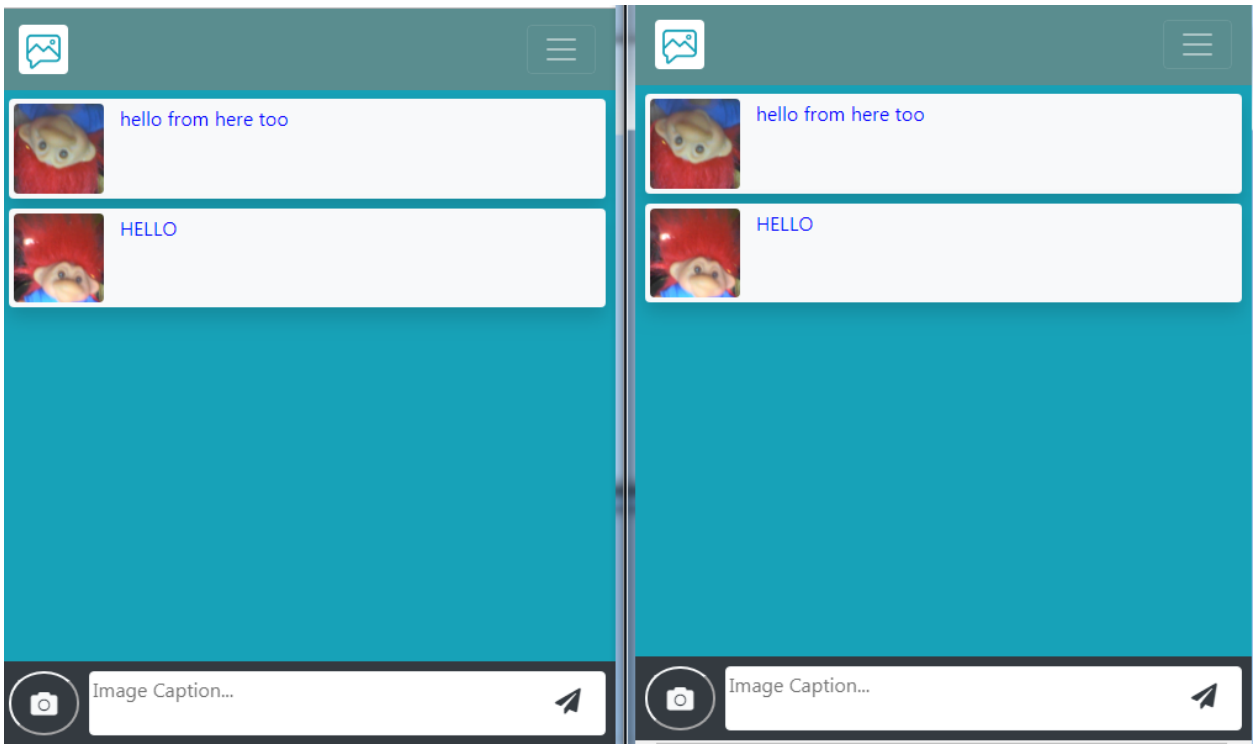
Τέλος, απαιτείται από τον socket server να προκαλεί το ‘all_messages’ event στον client οπότε στο server/index.js γίνεται προσθήκη πάνω από το //listen for new messages, η εντολή:

```
//Send all messages to connecting client
socket.emit(‘all_messages’, messages);
```

```
//Create web socket server
const io = socketio(server);

// Init server messages from disk
const messageData = fs.readFileSync(`${__dirname}/db.json`).toString();
const messages = messageData ? JSON.parse(messageData) : [];

// Listen for new socket client (connection)
io.on('connection', (socket) => {
    console.log("New Client Connected");

    //Send all messages to connecting client
    socket.emit('all_messages', messages);

    //listen for new messages
    socket.on('new_message', (message) => {

        //add to messages
        messages.unshift(message);

        //Persist to disk
        fs.writeFileSync(`${__dirname}/db.json`, JSON.stringify(messages));

        //broadcast new message to all connected clients
        socket.broadcast.emit('new_message', message);
    });
})

// Server "app" directory
app.use(express.static(`${__dirname}/../app`));
```



Βάζοντας μήνυμα και από τους δυο clients όλα τρέχουν κανονικά και τα μηνύματα αποθηκεύονται στο chat_history.json αρχείο του server.



Client 1 *Client 2*

9.8 Προσθήκη Service Worker

Δημιουργείται στον φάκελο app το αρχείο sw.js και καλείται στην HTML κάνοντας register τον service worker.

Στο σημείο που γίνεται ο έλεγχος του compatibility του media devices API, απαιτείται επίσης ο έλεγχος του compatibility για τον service worker. Κάτω από την _init() προστίθεται:

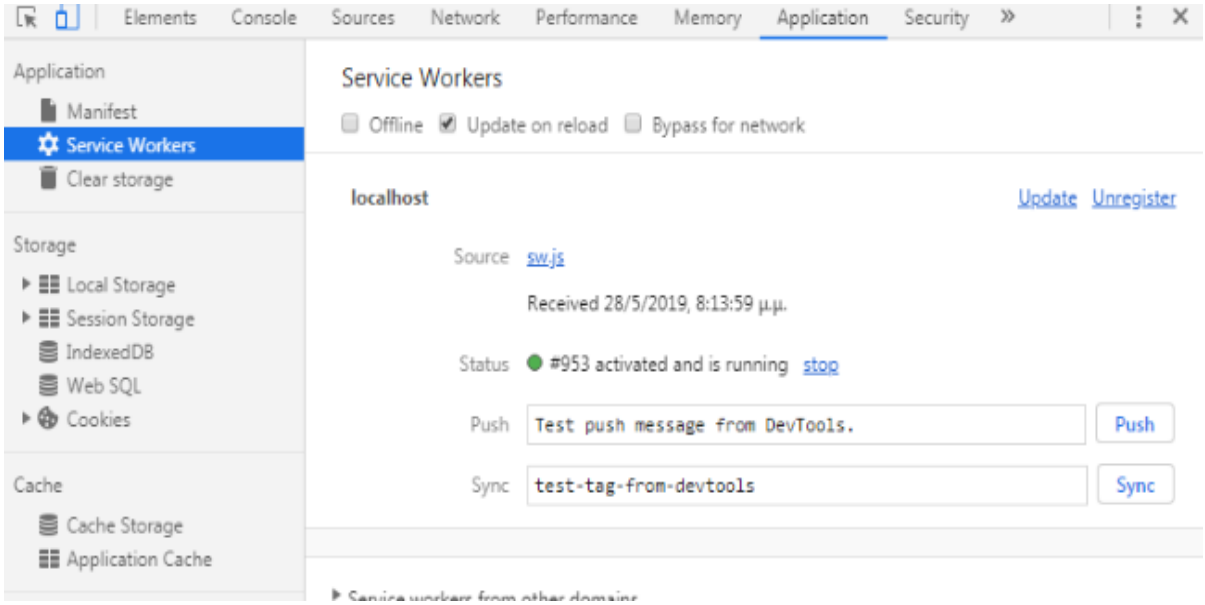
```
if ( 'serviceWorker' in navigator ){
    window.addEventListener('load', () => {
        navigator.serviceWorker.register('/sw.js');
    });
}
```

```
<!-- Init App if Supported -->
<script>

    // Check Compatibility
    if( 'mediaDevices' in navigator ){

        //call init() from main.js
        _init();

        // Register SW if supported
        if( 'serviceWorker' in navigator ){
            window.addEventListener('load', () => {
                navigator.serviceWorker.register('/sw.js');
            });
        }
```

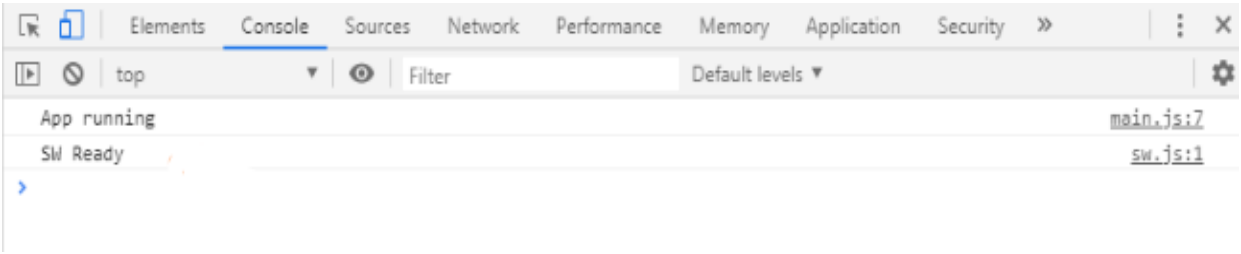
Μέσα στο sw.js προστίθεται ένα console.log() που δείχνει στην κονσόλα το μήνυμα “SW ready” για την επιβεβαίωση λειτουργίας του.

```
console.log('SW ready');
```

Μετά από μία επαναφόρτωση, είναι εμφανές στο application tab των devtools πως η εγγραφή του service worker έγινε επιτυχώς.



Παρομοίως γνωστοποιείται και απο την κονσολα.



Στη συνέχεια, μέσα στο sw.js αρχείο γίνεται δήλωση της cache έκδοσης ονομαζόμενη ως `static-${version}` όπου ${version} η τιμή που δηλώνεται και αφορά την έκδοση. Έπειτα, δηλώνονται τα αρχεία του app shell που χρειάζεται να αποθηκεύονται για την offline λειτουργία σε μία σταθερά appAssets. Η αποθήκευση των πόρων του app shell στην cache γίνεται κατά το install του service worker.

```
// Version
const version = '1.0';


// Static Cache - App Shell
const appAssets = [
    '/classes/message.js',
    '/classes/camera.js',
    '/room.html’,
    '/room.js'
];


// SW Install
self.addEventListener( 'install', e => {
    e.waitUntil(
        caches.open( `static-${version}` )
            .then( cache => cache.addAll(appAssets) )
    );
});
```

Όταν ο service worker ανανεώνεται απαιτείται η εκκαθάριση της cache και η εκ νέου αποθήκευση των appAssets.

```
// SW Activate
self.addEventListener( 'activate', e => {

    // Clean static cache
    let cleaned = caches.keys().then( keys => {
        keys.forEach( key => {
            if ( key !== `static-${version}` && key.match('static-') ) {
                return caches.delete(key);
            }
        });
    });
    e.waitUntil(cleaned);
});
```

Το επόμενο βήμα είναι η εισαγωγή ενός event listener στο fetch event και η φόρτωση του app shell μέσα από την cache. Τα icons που χρησιμοποιούνται είναι κατευθείαν από τον ιστότοπο του font-awesome, οπότε για να μπει ένας πόρος δικτύου στην cache θα συμπεριληφθεί το URL του για να υπάρξει πρόσβαση στα requests που κάνει ένας client στον font-awesome server. Χρησιμοποιούνται δύο διαφορετικές **caching στρατηγικές**.

```
// SW Fetch
self.addEventListener('fetch', (e) => {

    // App shell
    if( e.request.url.match(location.origin) ) {
        e.respondWith( staticCache(e.request) );

    // font-awesome
    } else if ( e.request.url.match('use.fontawesome.com') ) {
        e.respondWith( fallbackCache(e.request) );
    }
});
```

Η πρώτη στρατηγική είναι η staticCache(). Όταν ένας client εισέλθει στη σελίδα, γίνονται requests για πόρους που καλούνται μέσα από την HTML. Εάν κάποιο request για έναν πόρο που αντιστοιχεί στον φάκελο app είναι επιτυχημένο, θα γυρίσει επιτυχημένο response με τα αιτούμενα

δεδομένα από την cache. Εάν όμως το request αποτύχει θα τραβήξει από το δίκτυο το response, θα το αποθηκεύσει στην cache και θα σερβίρει στον client έναν response κλώνο.

Η δεύτερη είναι η fallbackCache() και αφορά μόνο τα εικονίδια που προέρχονται από τον ιστότοπο font-awesome.com. Αυτή η στρατηγική δοκιμάζει πάντα πρώτο το διαδίκτυο για την ανάκτηση πόρων και εάν αυτό αποτύχει ή δεν είναι διαθέσιμο, τρέχει στο τελευταίο παρόμοιο request που ήταν επιτυχές στην κρυφή μνήμη ως εναλλακτική λύση και το φορτώνει στον client. Είναι απαραίτητη η ενημέρωση της cache κάθε φορά που το request αυτό ολοκληρώνεται. Η λειτουργία επιστρέφει ένα fetch του αιτήματος έχοντας στη συνέχεια ένα response.

```
// Static cache strategy - Cache with Network Fallback
const staticCache = ( req, cacheName = `static-${version}` ) => {
    return caches.match(req).then( cachedRes => {
        // Return cached response if found
        if(cachedRes) return cachedRes;

        // Fall back to network
        return fetch(req).then ( networkRes => {
            // Update cache with new response
            caches.open(cacheName)
                .then( cache => cache.put( req, networkRes ));
            // Return Clone of Network Response
            return networkRes.clone();
        });
    });
};

// Network with Cache Fallback
const fallbackCache = (req, cacheName = `static-${version}`) => {
    // Try Network
    return fetch(req).then( networkRes => {
        // Check res is OK, else go to cache
        if( !networkRes.ok ) throw 'Fetch Error';
        // Update cache
        caches.open( cacheName )
```

```
            .then( cache => cache.put( req, networkRes ) );


        // Return Clone of Network Response
        return networkRes.clone();
    })
    // Try cache
    .catch( err => caches.match(req) );
};
```

Τρέχοντας την εφαρμογή, γίνεται εγκατάσταση του service worker . Είναι φανεροί όλοι οι επιλεγμένοι πόροι μέσα στην cache. Πηγαίνοντας offline και επαναφορτώνοντας τη σελίδα φορτώνεται το app shell και το toastr ενημερώνει πως το δίκτυο δεν είναι διαθέσιμο.

9.9 Προσθέτοντας Αυθεντικοποίηση Χρήστη

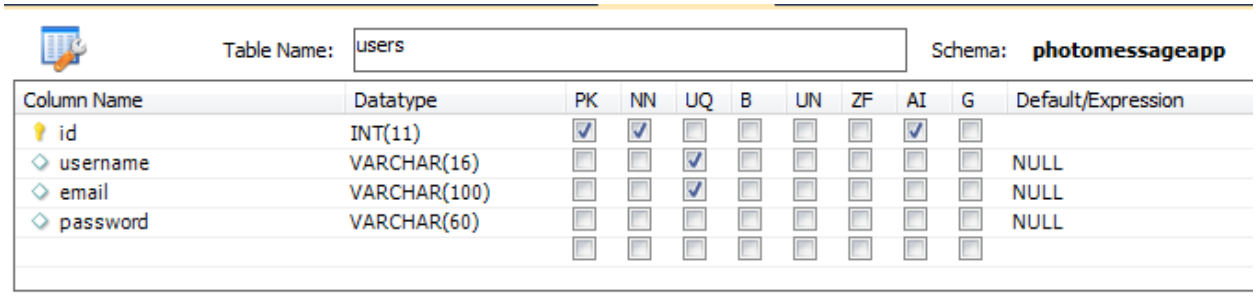
Παρακάτω υλοποιείται η προσθήκη ασφαλείας και η μετατροπή της εφαρμογής σε ένα user-based app απαιτώντας την αυθεντικοποίηση του εκάστοτε χρήστη που την επισκέπτεται. Ο κάθε χρήστης θα έχει το προσωπικό του χώρο, μέσα από την δυνατότητα να κάνει εγγραφή και στη συνέχεια να προχωρήσει στην εφαρμογή του chat room. Για να γίνει αυτό, χρειάζονται νέες σελίδες που αφορούν το login, το register και το profile του χρήστη. Επίσης, θα πρέπει οι σελίδες αυτές να συνδέονται έτσι ώστε να προστατεύουν την εφαρμογή και να μην επιτρέπουν την πρόσβαση σε χρήστες που δεν έχουν εξουσιοδοτηθεί να έχουν πρόσβαση ακόμη (μέχρι δηλαδή να πιστοποιηθεί η ταυτότητα χρήστη).

Ένας χρήστης εξουσιοδείται, εφ'όσον συμφώνησε να γνωστοποιείται από την εφαρμογή μέσα από ένα όνομα χρήστη και ένα συνθηματικό, να έχει πρόσβαση στον προσωπικό του χώρο και στην εφαρμογή καθώς και να αποσυνδεθεί εάν το επιθυμεί. Για έναν συνδεδεμένο χρήστη οπότε, στο interface της εφαρμογής, στη γραμμή πλοήγησης (<navbar>) χρειάζεται στη θέση των <a> links του login και του register να εμφανίζεται το logout. Η χρήση στη διάρκεια χρόνου που ένας χρήστης παραμένει συνδεδεμένος ονόμαζεται **session**. Το session ακυρώνεται αν σταματήσει ο διακομιστής να λειτουργεί ή εάν κάνει επανεκκίνηση ή στην περίπτωση που ο client προχωρήσει στο logout. Η ύπαρξη του χρήστη

συνδέεται με την ύπαρξη των στοιχείων του μέσα σε κάποια βάση δεδομένων. Κατά το login, ο χρήστης οφείλει να αποδεικνύει ότι τα στοιχεία που εισάγει στη φόρμα είναι οι ίδιες πληροφορίες, οι ίδιες τιμές που φιλοξενούνται σε κάποιες στήλες ενός πίνακα που αφορούν τα user credentials του.

9.9.1 Database Connection

Δημιουργείται μία βάση δεδομένων. Χρησιμοποιείται η δωρεάν πλατφόρμα MySQL Workbench και μέσα από τον MySQL server που εγκαθίσται, ρυθμίζεται να εκτελείται τοπικά στον localhost:3306. Δημιουργείται η βάση δεδομένων (schema) `photomessageapp` και μέσα του δημιουργείται ο νέος πίνακας `users` με στήλες το id (PRIMARY_KEY, NOT_NULL, AUTO_INCREMENT), το username (VARCHAR(16), UNIQUE), το email (VARCHAR(100), UNIQUE) και το password (VARCHAR(60)).



Εγκαθίσταται το module `mysql` στο back end με την εντολή `yarn add mysql` από τον terminal. Στο φάκελο του project προστίθεται ένας φάκελος db και μέσα δημιουργείται ένα νέο αρχείο db.js. Εκείνο αφορά τη σύνδεση της εφαρμογής με τη βάση. Μέσα στο αρχείο προστίθεται [τα στοιχεία βάσης είναι παραδειγματικά].

```
const mysql = require('mysql');

const connection = mysql.createConnection({

    host: 'localhost',

    user: 'root',

    password: '56eES9q9X5LgYxPKTCrvfs8Z',

    database: 'photomessageapp'

});

connection.connect();

//test provided by https://www.npmjs.com/package/mysql

connection.query('SELECT 1 + 1 AS solution', function(error, results, fields){
```

```
    if (error) throw error;
    //console.log('The solution is: ', results[0].solution);
    console.log("DB Connected");
});
module.exports = connection;
```

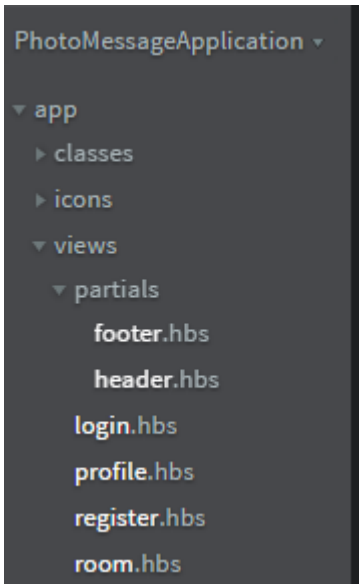
Το αρχείο αυτό καλείται μέσα στο server/index.js καλώντας το αρχείο με την require και αποθηκεύοντας το στη μεταβλητή db.

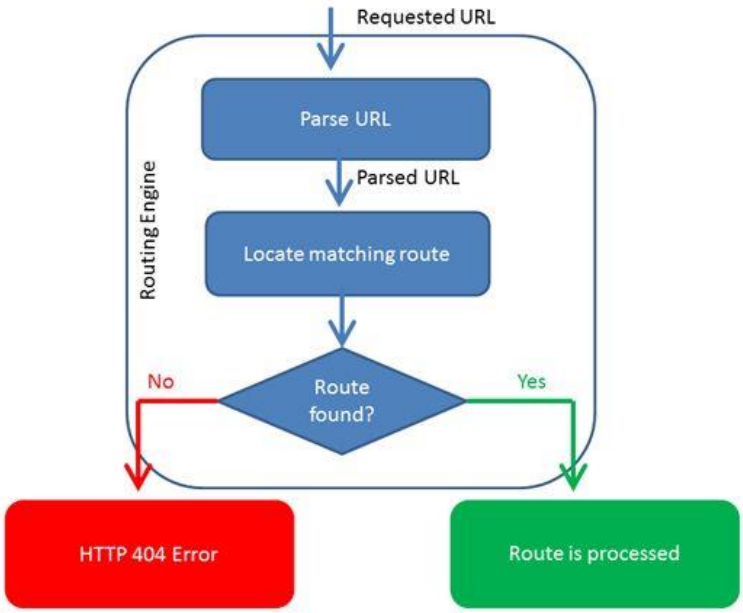
```
const db = require(`${__dirname}/../db/db.js`);
```

9.9.2 HBS - View Engine for Handlebars.js

Πριν τη δημιουργία της register σελίδας και εκτιμώντας πως στη συνέχεια πρόκειται να υπάρχουν περισσότερες σελίδες και λειτουργίες που θα απαρτίζουν το app, ο προγραμματισμός για την πλοήγηση αλλά και για τη μεταφορά δεδομένων κατά την πλοήγηση από σελίδα σε σελίδα είναι απαραίτητος και απαιτητικός. Προκειμένου να αποφευχθεί η πιθανή σύγχυση με τις λειτουργίες που τρέχουν στο back end, εγκαθίσταται στο project το hbs module που υποστηρίζεται από την Express και που χειραγωγεί και αντιμετωπίζει τα αρχεία .hbs (στην ουσία είναι αρχεία HTML που χειρίζονται την handlebars.js βιβλιοθήκη) σαν templates και που καθιστώνται (render) στο UI με συγκεκριμένα δεδομένα που προσφέρονται από το back-end. Κάθε σελίδα αντιστοιχεί σε ένα endpoint του URL και θα είναι προσβάσιμη από το χρήστη μέσα από τον browser.

Η διαδικασία χαρτογράφησης των endpoints σε ένα application λέγεται **routing** (ανήκει στο μοντέλο MVC). Τα endpoints λέγονται routes («μονοπάτια») και είναι απαραίτητη η δήλωση τους μέσα στον Express server. Θα μπορούσε να χρησιμοποιηθεί ο router που προσφέρεται απο το express-router module όμως λόγω των μικρών απαιτήσεων δεν χρειάζεται η χρήση του. Για παράδειγμα, όταν ο χρήστης πλοηγείται στην εφαρμογή και επισκευθεί το '/profile' route θα πραγματοποιήσει ένα GET request. Όμως για λόγους προστασίας, σε μη-εξουσιοδοτημένους χρήστες το GET request πρέπει να μην κάνει render το view αφού δεν έχουν πιστοποιηθεί. Μέσα από την hbs γίνεται εύκολη χειραγώγηση των GET και POST requests στα endpoints καθώς και στο routing. Επίσης, έχει ενσωματωμένη τη βιβλιοθήκη handlebars.js που επιτρέπει την δήλωση των {{ }} expressions μέσα στα αρχεία .hbs για δυναμικές μεταβλητές κατά το GET και POST.





Μέσα από τον terminal εκτελείται η εντολή `yarn add hbs`.

Μετονομάζεται το αρχείο *room.html* σε *room.hbs* και μεταφέρεται σε έναν νέο φάκελο /app/views. Μέσα στο /views δημιουργείται ο φάκελος /partials που φιλοξενεί τα κοινά αρχεία για όλα τα views, header.hbs και footer.hbs. Για την πλοήγηση του χρήστη είναι χρήσιμο ένα navigation bar που προσφέρει έτοιμο η Bootstrap στον ιστότοπο της. Επιλέγεται από το room.hbs το <head> και η γραμμή κώδικα <body class="bg-info"> καθώς και οι δηλώσεις στο τέλος του αρχείου:

```
<script src="/modules/jquery/dist/jquery.min.js"></script>

<script src="/modules/bootstrap/dist/js/bootstrap.min.js"></script>

<script src="/modules/toastr/build/toastr.min.js"></script>
```

Γίνονται αποκοπή. Στη συνέχεια, επικόλληση μέσα στο header.hbs. Από κάτω προστίθεται το navbar. Δημιουργούνται τα αρχεία login.hbs, profile.hbs και register.hbs μέσα στο φάκελο /views.

Ο κώδικας του navbar θα προσαρμοστεί αφού ο χρήστης αυθεντικοποιηθεί ώστε να εμφανίζει logout αντι login και profile αντι register.

Ο περιεχόμενο του /views/partials/header.hbs είναι:

```
<html lang="en">

<head>

    <title>Photo Message</title>

    <meta name="viewport" content="width=device-width, initial-scale=1">

    <meta name="mobile-web-app-capable" content="yes">

    <meta name="theme-color" content="#19A2B8">

    <meta http-equiv="cache-control" content="private, max-age=0, no-cache">

    <link rel="icon" href="/icons/favicon.ico" type="image/x-icon">

    <link rel="manifest" href="manifest.json">


    <!-- Styles -->

    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.8.2/css/all.css" integrity="sha384-
oS3vJWv+0UjzBfQzYUhtDYW+Pj2yciDJxpsK1OYPAYjqT085Qq/1cq5FLXAZQ7Ay" crossorigin="anonymous">

    <link rel="stylesheet" href="/modules/bootstrap/dist/css/bootstrap.min.css">

    <link rel="stylesheet" href="/modules/toastr/build/toastr.min.css">

    <style>

        #toast-container {

            top: 0;

            left: 0;
```

```
            width: 100%;
        }
        #toast-container > div {
            opacity: 1;
            box-shadow: 0 0 2rem rgba(0,0,0,0.75);
            border-radius: 0;
            width: 100%;
            max-width: 100%;
        }
        #camera.withphoto{
            color: transparent !important;
            background-size: cover;
        }
        h1,h2,h3,h4,h5,label,p {
            color:white;
        }
        .container{
            padding-top:80px;
        }
        .navbar{
            background-color: #5a8d8f !important;
        }
    </style>

    <!-- Scripts -->
    <script src="/modules/jquery/dist/jquery.min.js"></script>
    <script src="/modules/bootstrap/dist/js/bootstrap.min.js"></script>
    <script src="/modules/toastr/build/toastr.min.js"></script>
</head>
<!-- App Shell -->
```

```
<body class="bg-info">


    <nav class="navbar navbar-expand-md navbar-dark fixed-top">
        {{#if isAuthenticated}}
            <a class="navbar-brand" href="/">
                <img src='../icons/icon-128x128.png' width='40' />
            </a>
        {{else}}
            <a class="navbar-brand" href="/login">
                <img src='../icons/icon-128x128.png' width='40' />
            </a>
        {{/if}}
        <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarsExampleDefault" aria-
controls="navbarsExampleDefault" aria-expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>


        <div class="collapse navbar-collapse" id="navbarsExampleDefault">
            <ul class="navbar-nav mr-auto">


                {{#if isAuthenticated}}
                    <li class="nav-item active">
                        <a class="nav-link" href="/">Room</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="/profile">Profile</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="/logout">Logout</a>
                    </li>
```

```
            {{else}}
                <li class="nav-item active">
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="/register">Register</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="/login">Login</a>
                </li>
            {{/if}}
        </ul>
    </div>
</nav>
<script>
    // Dynamically change .nav > a active class on view rendering
    $(document).ready(function () {
        const pageURL = '/' + $(location).attr("href").split("/").pop();
        console.log(pageURL);
        $('.active').removeClass('active');
        $('a[href$="'+ pageURL +'"]').parent().addClass('active');
    });
</script>
```

Αντίστοιχα το περιεχόμενο του /views/partials/footer.hbs:

```
</body>
<!-- App Shell -->
</html>
```

9.9.3 Register

Το περιεχόμενο του register.hbs view είναι:

```
{{> header}}

        <div class="container">

        <h1>{{title}}</h1>

        <h5 style='color:#b93b3b;'>

            {{subTitle}}

        </h5>

        <!--passing error messages from backend using hsb-->

        <div id='hiddenErrors' hidden>

        {{#if errors}}

            {{#each errors}}

                <p class='errorMsg'>

                    {{this.msg}}

                <p/>

            {{/each}}

        {{/if}}

        </div>


        <form id='registerForm' method="POST" action='/register' {{formVisible}}>

            <div class="form-group">

                <label for="username">Username</label>

                <input type='text' class='form-control' name='username' placeholder='Username'>

            </div>

            <div class="form-group">

                <label for="email">Email Address</label>

                <input type='email' class='form-control' name='email' placeholder='Email'>

            </div>

            <div class="form-group">

                <label for="password">Password</label>
```

```
                <input type='password' class='form-control' name='password' placeholder='Password'>
            </div>
            <div class="form-group">
                <label for="passwordMatch">Re-Enter Password</label>
                <input type='password' class='form-control' name='passwordMatch' placeholder='Password'>
            </div>
            <button type='submit' class='btn btn-light'>Submit</button>
        </form>

        <div id='redirect' style='display:none;color:white;'><p>Redirecting</p></div>
    </div>

<!--
    <script src="/modules/jquery/dist/jquery.min.js"></script>
    <script src="/modules/bootstrap/dist/js/bootstrap.min.js"></script>
    <script src="/modules/toastr/build/toastr.min.js"></script>
-->
    <script>
        $(document).ready(function(){
            console.log("register page rendered");
            //Detect if there are any errors and display them in a toastr
            var count = $("#hiddenErrors").find("p").length;
            console.log("error messages:" + count);
            if ( count > 0 ) {
                $('.errorMsg').each(function(i, obj){
                    var errorMessage = $(this).html();
                    toastr.error( null, errorMessage, {timeout: 3000});
                });
            }
```

```
            //Detect if form is hidden => redirect('/')
            if($("#registerForm").is(":hidden")){
                console.log('hidden');
                $('#redirect').show();
                setTimeout(function () {
                    // after 2 seconds
                    window.location = "/";
                }, 2000);
            }
        });
    </script>
{{> footer}}
```

Μέσα στο server/index.js δηλώνεται το view engine:

```
// router rendering looks up for a views file
// specify view AND partial folders and parse the
// engine to hbs.
app.set('views',`${__dirname}/../app/views`);
hbs.registerPartials(`${__dirname}/../app/views/partials`);
app.set('view engine', 'hbs');
```

Τα στοιχεία της φόρμας επαληθεύονται μέσα από τον server οπότε χρειάζεται ένας τρόπος να περνάνε τα δεδομένα στο server/index.js. Αυτό γίνεται δυνατό με το body-parser module που θα πιάνει τα values των inputs μέσα από τη φόρμα του /register επιλέγοντας το name attribute των input στοιχείων. Επίσης, γίνεται εγκατάσταση του express-validator module που επιτρέπει τον έλεγχο των δεδομένων. Μετά την εγκατάσταση μέσα από τις εντολές `yarn add body-parser`, `yarn add express-validator` δηλώνονται τα αρχεία στην αρχή του index.js.

```
const bodyParser = require('body-parser');
const expressValidator = require('express-validator');
```

Κάτω από το κομμάτι του κώδικα που δημιουργείται ο socket server, προστίθεται:

```
// INITIALIZE BODYPARSER

app.use(bodyParser.urlencoded({extended: false}));

app.use(bodyParser.json());


// From express-validator docs, the method

// app.use(expressValidator) must be immediately after

// any of the bodyParser middlewares

app.use(expressValidator());
```

Και κάτω από τη δήλωση του view engine γίνεται εστίαση στο routing και προστίθεται ο παρακάτω κώδικας που αφορά το GET request στο /register endpoint (όταν ο χρήστης δηλαδή μπαίνει στο register view) καθώς και ο κώδικας για το POST request (όταν ο χρήστης πατήσει submit).

```
//AUTH: WHENEVER CLIENT ENTERS REGISTRATION

app.get('/register', function (req, res) {

  res.render('register', {

      title: 'Registration',

      formVisible: 'visible'

  });

});


//AUTH: REGISTER ON SUBMIT

app.post('/register',function (req, res) {

    // Validate inputs for accurate data

    // Also the fields username and email in DB must get setted to UNIQUE key type.

    req.checkBody('username', 'Username field cannot be empty.').notEmpty();

    req.checkBody('username', 'Username must be between 4-15 characters long.').len(4,
15);

    req.checkBody('email', 'The email you entered is invalid, please try
again.').isEmail();
```

```
    req.checkBody('email', 'Email address must be between 4-100 characters long, please
try again.').len(4, 100);

    req.checkBody('password', 'Password must be between 8-100 characters long.').len(8,
100);

    req.checkBody("password", "Password must include one lowercase character, one
uppercase character, a number, and a special character.").matches(/^(?=.*\d)(?=.*[a-
z])(?=.*[A-Z])(?!.* )(?=.*[^a-zA-Z0-9]).{8,}$/, "i");

    req.checkBody('passwordMatch', 'Password must be between 8-100 characters
long.').len(8, 100);

    req.checkBody('passwordMatch', 'Passwords do not match, please try
again.').equals(req.body.password);

    // Additional validation to ensure username is alphanumeric with underscores and
dashes

    req.checkBody('username', 'Username can only contain letters, numbers, or
underscores.').matches(/^[A-Za-z0-9_-]+$/, 'i');


    const errors = req.validationErrors();


    if(errors) {

        console.log(`errors: ${JSON.stringify(errors)}`);

        res.render('register',{

            title: 'Registration Error',

            subTitle: 'please try again.',

            formVisible: 'visible',

            errors: errors

        });

    }

});
```

Το validation γίνεται μέσα από RegExps(regular expressions) που αφορούν τα όρια του κάθε input στη register φόρμα και σε περίπτωση που υπάρχουν errors, ομαδοποιούνται και εμφανίζονται στο χρήστη μέσα από ξεχωριστά toastr για κάθε error. Στην περίπτωση που δεν υπάρχουν errors θα υπάρχει ένα else statement στο τέλος της “if(errors){“ ώστε να γίνει η αποθήκευση των δεδομένων του νέου χρήστη στη βάση δεδομένων. Το password θα κωδικοποιείται μέσα στη βάση για λόγους ασφαλείας. Για αυτό το σκοπό εγκαθίσταται το bcrypt module που επιτρέπει εύκολο hashing σε

passwords. Απαιτεί να δηλωθεί μία μεταβλητή `saltRounds` που αφορά τον αριθμό των κύκλων κρυπτογράφησης που θα περάσει το password.

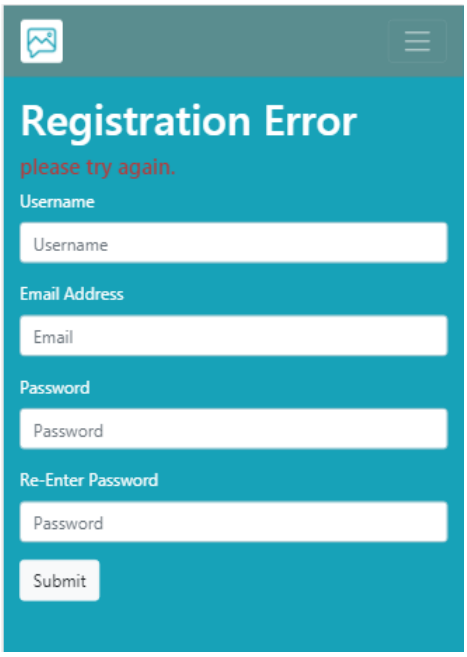
Στο τέλος των δηλώσεων των modules του server στο index.js δηλώνεται:

```
const saltRounds = 10;

const bcrypt = require('bcrypt');

var user_obj = {};
```

Μέσα στο POST request της /register, στην περίπτωση που δεν υπάρχουν errors, περνάνε τα values των inputs, το password κωδικοποιείται και τρέχει ένα INSERT query στη βάση.

```
    } else {
        // Access form inputs with bodyParser
        const username = req.body.username;
        const email = req.body.email;
        const password = req.body.password;

        // Hash password
        bcrypt.hash(password, saltRounds, (err,hash) => {
            // Insert Query
            // using ? as value and using [] as the second argument
            // to make connection escape users' values automatically and prevent
malicious data
            db.query('INSERT INTO users (username,email,password) VALUES (?, ?, ?)',
[username, email, hash], (error, results, fields) => {
                if (error) throw error;
                db.query('SELECT LAST_INSERT_ID() as user_id',(error, results, fields)
=> {
                  if(error) throw error;
                  user_obj = results[0];
                });
            });
        });
```

```
}
```



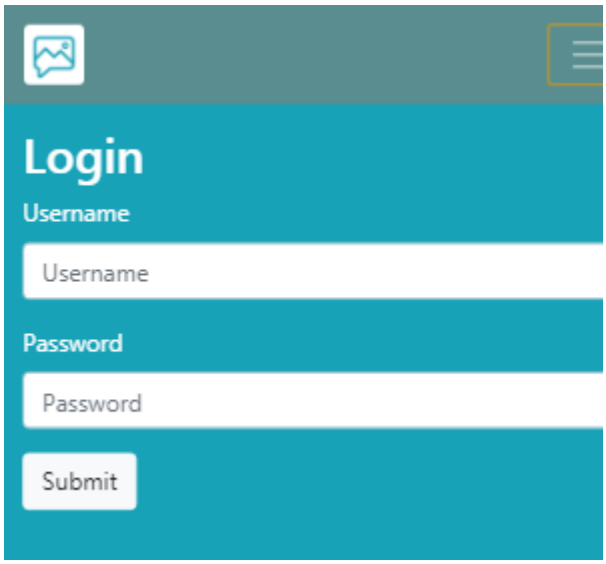
Αφού τα στοιχεία εισαχθούν στη βάση, εκτελείται άλλο ένα query που επιλέγει το τελευταίο εισαγόμενο id και το αποθηκεύει στη μεταβλητή user_obj. Έπειτα, φορτώνεται το register view με τη φόρμα να είναι hidden και μέσα από ένα script που υπάρχει στο register.hbs γίνεται redirect μετά από 2 δευτερόλεπτα στο /login view.

9.9.4 Login (Update User Session, Return Auth Cookie)

Στο αρχείο login.hbs προστίθεται το παρακάτω περιεχόμενο:

```
{{> header}}

<div class='container'>
        <h2>{{title}}</h2>
        <form id='loginForm' method="POST" action='/login' {{formVisible}}>
            <div class="form-group">
                <label for="username">Username</label>
                <input type='text' class='form-control' name='username' placeholder='Username'>
            </div>
            <div class="form-group">
                <label for="password">Password</label>
                <input type='password' class='form-control' name='password' placeholder='Password'>
            </div>
            <button type='submit' class='btn btn-light'>Submit</button>
        </form>
</div>

<script>
  // Check Compatibility
        if( 'mediaDevices' in navigator ){
```

```
            // Register SW if supported
            if( 'serviceWorker' in navigator ){
                window.addEventListener('load', () => {
                    navigator.serviceWorker.register('/sw.js');
                });
            }
        } else {
            console.log("error");
            // Show Notification
            toastr.error( null, "This App is not supported by your browser.", {timeout: 3000});
        }
</script>
{{> footer}}
```



Στη σελίδα του /login υπάρχει η φόρμα με τα δύο inputs, username και password, καθώς και το submit button. Συμπληρώνοντας τη φόρμα και πατώντας το submit πραγματοποιείται ένα POST request στο /login route.

Όταν το login γίνει επιτυχώς και τα στοιχεία του login form ταυτίζονται με τα στοιχεία της βάσης, η εφαρμογή θα πρέπει να αναγνωρίζει τον user κατά τη διάρκεια του session. Για να το πετύχει αυτό, εγκαθίσταται το express-session module που θα δημιουργήσει αυτόματα ένα session cookie στον browser και θα αποθηκεύει την τιμή του authentication (boolean) καθώς εγκαθίσταται και το MySQLStore που είναι απαραίτητο από το express-session module αφού δημιουργεί έναν πίνακα στη βάση με σκοπό την καταγραφή των sessions που πραγματοποιούνται. Έπειτα γίνεται εγκατάσταση το passportjs που αφορά το authentication καθώς και το passport-local που είναι κομμάτι του passport και λειτουργεί μέσα από μεθόδους-στρατηγικές.

```
const session = require('express-session');
const MySQLStore = require('express-mysql-session') (session);
const passport = require('passport');
```

```
const LocalStrategy = require('passport-local').Strategy;
```

Στο index.js πρέπει να ρυθμιστεί το sessionStore και το passport.js καθώς και να ανανεώνεται η τιμή που επιστρέφεται από την res.Locals.isAuthenticated() και που χρειάζεται το LocalStrategy. Προστίθεται κάτω από το var user_obj = {} ο παρακάτω κώδικας:

```
const options = {
    host: 'localhost',
    user: 'root',
    password: '56eES9q9X5LgYxPKTCrvfs8Z',
    database: 'photomessageapp'
}

//Define sessionStore, if also sessions table doesn’t exist, create it automatically
const sessionStore = new MySQLStore(options);

app.use(session({
//    store: sessionStore,
    secret: 'iowjcxz3ivjewqn',
    resave: false,
    saveUninitialized: true
}));



app.use(passport.initialize());
app.use(passport.session());

// Get locals for strategy
app.use(function(req,res,next){
    res.locals.isAuthenticated = req.isAuthenticated();
    next();
})
```

```
//Strategy for /login POST action

passport.use(new LocalStrategy( function(username, password, done) {

    db.query('SELECT password FROM users WHERE username = ?', [username], (err, results,
fields) => {

        if (err) {done(err)};


        if (results.length === 0){

            done(null, false);

        } else {

            const hash = results[0].password.toString();

            console.log("hash:"+hash);


            bcrypt.compare(password, hash, function(err, response) {

                if (response === true) {

                    console.log(results);

                    return done(null, {user_id: results[0].id, userName: username});

                } else {

                    return done(null, false);

                }

            });

            }

    });

}));
```

Η παραπάνω στρατηγική θα εφαρμοστεί κατά το POST request στο /login και θα συγκρίνει το password value του input με το password κελί του πίνακα “users” στη βάση αφού το αποκρυπτογραφήσει. Το done() που επιστρέφεται αφορά το promise που θα γίνει resolved στην εκάστοτε περίπτωση και ανανεώνει το user_obj.

Για να πετύχει όμως το passport την ολοκληρωμένη λειτουργία του θα πρέπει να προστεθεί στο server/index.js πάνω από το // Server ‘’app’’ folder:

```
passport.serializeUser((user_id, done) => {

   done(null, user_id);
```

```
});


passport.deserializeUser((user_id, done) => {
    done(null, user_id);
});
```

Το serializeUser παίρνει θέση όταν ο χρήστης πιστοποιείται και ανανεώνεται το session. Το deserializeUser αντίστοιχα αφορά την έξοδο του χρήστη από το session. Οι δυο αυτές λειτουργίες είναι απαραίτητες για να δουλέψει σωστά το passport. Έπειτα, κάτω από το // Server "node_modules" directory προστίθεται:

```
//Determine if user is logged in
function authenticationMiddleware() {
    return (req, res, next) => {
        if(req.isAuthenticated() === true){
            console.log("authenticated!");
            return next();
        } else {
            console.log("not authenticated");
            res.redirect('/login');
        }
    }
}
```

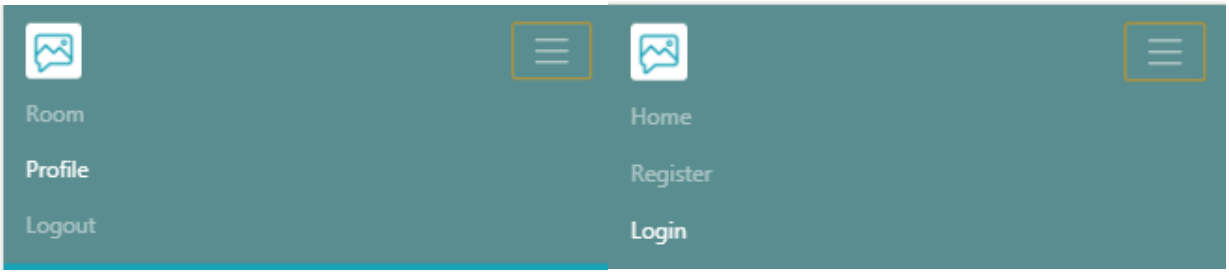
Με την παραπάνω λειτουργία γίνεται έλεγχος εάν είναι συνδεδεμένος ο user κάθε φορά που γίνεται κάποιο request κατά την πλοήγηση του στα /views. Εάν ναι, τότε επιστρέφει τη next() μέθοδο η οποία λέει στη javascript να βγει από τη μέθοδο που βρίσκεται την προκειμένη στιγμή και να συνεχίσει να τρέχει το πρόγραμμα στο αρχείο έξω από αυτή. Η next χρησιμοποιείται ως μεσάζοντας και συγκεκριμένα ως δεύτερη παράμετρος κατά το routing. Αναλύεται πρακτικά στη συνέχεια.

Για το login και τα GET,POST requests που γίνονται στο route του /views/login.hbs. προστίθεται κάτω από το routing του register ο κώδικας:

```
//AUTH: WHENEVER CLIENT ENTERS LOGIN
app.get('/login', function(req, res, next) {
    console.log("user auth: " + req.isAuthenticated());
```

```
    if( req.isAuthenticated() === true ){
        res.redirect('/profile');
    } else {
        res.render('login',{
            title: 'Login'
        });
    }
});
//AUTH: WHENEVER CLIENT SUBMITS LOGIN FORM
app.post('/login', passport.authenticate('local', {
    successRedirect: '/profile',
    failureRedirect: '/login'
}));
```

Κατά το GET του /login τσεκάρεται το session και εάν είναι authenticated ο user, τότε πραγματοποιείται redirect στο /profile view. Κατά το POST request του /login ενεργοποιείται η στρατηγική της passport και εάν είναι επιτυχημένη αυθεντικοποίηση, προχωράει στο /profile view, αλλιώς φορτώνεται ξανά το /login.



Header authorized *Header unauthorized*

9.9.5 Profile View Και Logout

Το profile είναι το view που συναντά ο χρήστης κατά την προσέλευση του στον ιστότοπο εάν είναι πιστοποιημένος. Στο αρχείο /views/profile.hbs προστίθεται το παρακάτω περιεχόμενο:

```
{{> header}}
<div class='container'>
    <h2>Welcome {{currentUser}}</h2>
        <hr>
```

```
        <h4>Chat Options</h4>
        <form id='profileForm' method="POST" action='/profile'>
            <div class='form-group'>
                <label for='nameColor'>Name Color</label>
                <input id='nameColor'
                name='nameColor'
                class='form-control'
                value='{{nameColor}}'
                type='color'>
            </div>

            <div class='form-group'>
                <label for='msgColor'>Message Color</label>
                <input id='msgColor' name='msgColor'    class='form-control' value='{{msgColor}}'
type='color'>
            </div>

            <button type='submit' class='btn btn-light'>Save</button>
        </form>
{{> footer}}
```

Το view περιμένει μία μεταβλητή currentUser που περνάει μέσα από το session και καλωσορίζει τον χρήστη με το username του αναλόγως. Ο χρήστης μπορεί να επιλέξει χρώμα για το όνομα και χρώμα για το κείμενο που θα αποθηκευτούν στην κλάση των messages κατα το send(). Ανανεώνεται στο classes/message.js η add(data_url, caption_text) σε:

```
//Add a new message
    add(user_name, data_url, caption_text, name_color, msgColor) {

        //create message obj
```

```
        let message = {
            username: user_name,
            photo : data_url,
            caption : caption_text,
            nameColor: name_color,
            msgColor: msgColor
        }

         // add to local messages
        this.messages.unshift(message);

        //emit to server
        console.log('sending to server');
        this.socket.emit('new_message', message);

        //return formatted message obj
        return message;
    }
}
```

Στο room.hbs πάνω από το <main> προστίθεται:

```
<div hidden id="userName">{{username}}</div>
<div hidden id='nameColor'>{{nameColor}}</div>
<div hidden id='msgColor'>{{msgColor}}</div>
```

Μέσα στο script του room.hbs ανανεώνεται η κλήση της _init() σε:

```
var username = $('#userName').text();
var nameColor = $('#nameColor').text();
var msgColor = $('#msgColor').text();
```

```
    //call init() from room.js
    _init(username, nameColor, msgColor);
```

Τέλος, προστίθεται το styling στο τέλος του αρχείου.

```
<style>
        .names{
            font-weight: 900;
            text-align:left;
            width: 100%;
            margin: 0px 20px;
        }
        .messages{
            text-align: left;
            width: 100%;
            margin: 0px 20px;
        }
</style>
```

Στο αρχείο room.js ανανεώνεται η send.onClick μέσα στην _init():

```
  //Submit Photo
    $("#send").on("click", () => {

        //Get caption text
        let caption = $("#caption").val();

        //Check message if ok
        if(!camera.photo || !caption) {
```

```
            //Show notification and return
            toastr.warning('Photo & Caption Required.', 'Incomplete Message');
            return; }


        let message = messages.add (userName, camera.photo, caption, nameColor,
msgColor);


        console.log(messages.all);


        // Render new message in feed
        renderMessage( message );


        // Reset caption field on success
        $('#caption').val('');
               $('#camera').css('background-image','').removeClass('withphoto');
        camera.photo = null;
    });
```

Και παρακάτω ανανεώνεται η renderMessage() σε:

```
// Create new message element
const renderMessage = (message) => {
    // Message HTML
    let msgHtml = `<div style="display:none;" class="row message bg-light mb-2 rounded
shadow">
                        <div class="col-2 p-1">
                            <img src="${message.photo}" class="photo w-100 rounded">
                        </div>
                        <div class="col-10 p-1">
                            <div class='row'>
                            <p class='names'
style='color:${message.nameColor}'>${message.username}</p>
```

```
                                </div>
                                <div class='row'>
                                <p class='messages'
style='color:${message.msgColor}'>${message.caption}</p>
                                </div>
                            </div>
                        </div>`;

    $(msgHtml).prependTo('#messages').show(500)

    //bind a click handler on new img element to show in modal
    .find('img').on("click",showPhoto);
};
```

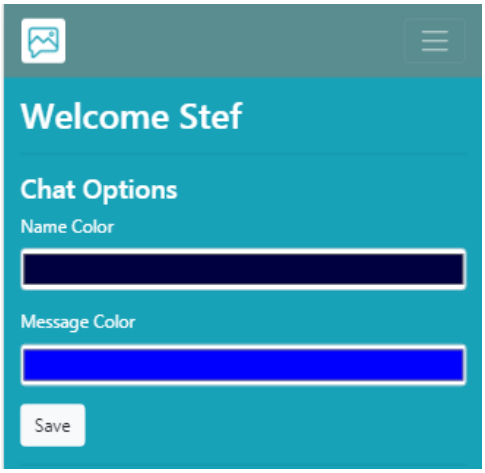
Για το routing του /profile προστίθεται στο server/index.js:

```
//AUTH: WHENEVER CLIENT ENTERS PROFILE
app.get('/profile', authenticationMiddleware(), function(req, res, next) {
    console.log("user auth: " + req.isAuthenticated());
    if( req.isAuthenticated() === true ){
        console.log(req.session.passport.user);
        db.query('SELECT id FROM users WHERE username = ?',
[req.session.passport.user.userName], (error, results, fields) => {
            if (error) throw error;
            req.session.passport.user.user_id = results[0].id;
            console.log("passport user: ",req.session.passport.user);
            db.query('SELECT * FROM profile_options WHERE user_id = ?', [results[0].id],
(error, results2, fields) => {
                if (error) throw error;

                console.log(results2);

                res.render('profile',{
```

```
                    currentUser: req.session.passport.user.userName,
                    nameColor: results2[0].name_color,
                    msgColor: results2[0].message_color
                });
            });
        });
    }
});
app.post('/profile', function(req, res, next) {
//    if( req.isAuthenticated() === true ){
        var nameColor = req.body.nameColor;
        var msgColor = req.body.msgColor;
        var userId = req.session.passport.user.user_id;
        db.query('UPDATE profile_options SET name_color = "' + nameColor + '",
message_color = "' + msgColor + '" WHERE user_id = "'+userId+'" ', (error, results,
fields) => {
            if (error) throw error;
            res.render('profile',{
                currentUser : req.session.passport.user.userName,
                nameColor: nameColor,
                msgColor: msgColor
            });
        });
```

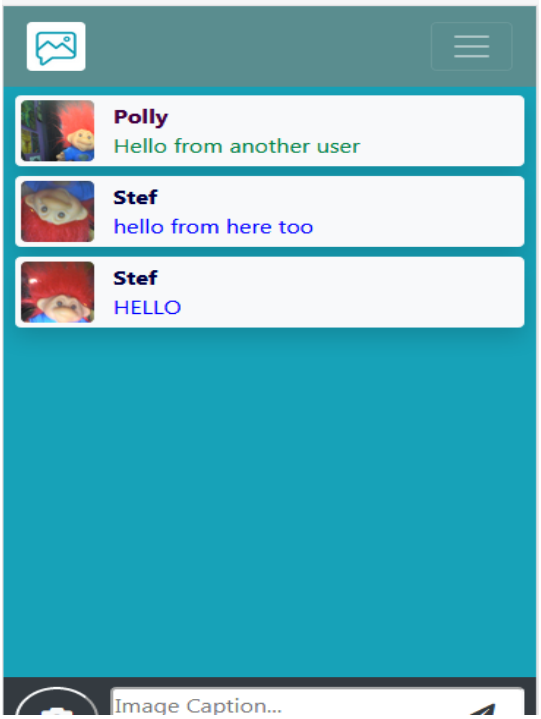
```
//     }
});
```

Αν ο χρήστης μπει στο profile και δεν είναι authenticated τότε γυρνάει στο /login χάρις την authenticationMiddleware() μέθοδο που αναφέρθηκε προηγουμένως και που δηλώθηκε στο τέλος του index.js.



Profile view, δεξιά εικόνα.

Με αυτό τον τρόπο στο room.hbs και στα messages εμφανίζονται τα ονόματα των χρηστών με το αντίστοιχο message που έστειλε ο κάθε χρήστης, αποθηκεύονται στο chat_history.json αρκεί να διαγραφούν τα προηγούμενα δεδομένα που δεν περιέχουν nameColor και msgColor χαρακτηριστικά.



Το επόμενο που πρέπει να κατασκευαστεί είναι το /logout route που θα εμφανίζεται στο navbar κατα το session κάποιου συνδεδεμένου χρήστη.

Στο server/index.js και μέσα στο routing προστίθεται:

```
app.get('/logout', function(req, res){
    req.logout();
    req.session.destroy();
    user_obj = {};
    res.redirect('/login');
});
```

Το passportjs σε συνδυασμό με το express-session κάνει τη διαδικασία του authentication πολύ εύκολη ανανεώνοντας δυναμικά τα tokens του cookie και το sessionStore στη βάση δεδομένων.

9.10 Push Notifications

Δημιουργείται ένας Push διακομιστής και ένα αρχείο που διαχειρίζεται την συνδρομή push. Ο Push server είναι κατασκευασμένος με Node.js χρησιμοποιώντας το web-push module. Πρόκειται για

έναν HTTP server ικανό να παρέχει ένα δημόσιο κλειδί στον client, εκείνος να δέχεται συνδρομές push και ο push server να στέλνει προσαρμοσμένα push notifications σε όλους τους *εγγεγραμμένους* πελάτες.
Ο client θα μπορεί να κάνει εγγραφή στη συνδρομή και κατάργηση της μέσα από το profile view. Προστίθεται κάτω από την φόρμα των colors ο εξής κώδικας:

```
        <hr>
          <h4>Push Notifications</h4>
          <div id="subscribe" style="text-align: center;">
              <p><i>Subscribe to receive Push Notifications</i></p>
              <button onclick="subscribe()">Subscribe</button>
          </div>
          <div id="unsubscribe" class="hidden" style="text-align: center;">
              <p><i>Unsubscribe from Push Notifications</i></p>
              <button onclick="unsubscribe()">Unsubscribe</button>
          </div>
<style>
button {
      padding: 0.5rem 3rem;
      font-size: 2rem;
      background: #66DE93;
      color: white;
      border: none;
      border-radius: 5px;
   }
   #unsubscribe button {
      background: #EB3F2F;
   }
   .hidden {
      display: none;
   }
   input[type="color"] {
      padding: 0 !important;
   }
</style>
```

Στο view του /profile τώρα θα εμφανίζεται το κουμπί Subscribe και το Unsubscribe με το δεύτερο να είναι κρυμμένο. Ο κάθε client έχει τη δυνατότητα να εγγραφεί (αν δεν έχει ήδη εγγραφεί) και να ακυρώσει την εγγραφή όποτε το επιθυμεί. Δημιουργούνται δύο τμήματα για το κάθε κουμπί τα οποία θα είναι ορατά ανάλογα το εάν υπάρχει συνδρομή push. Επίσης, έγινε προσθήκη css styling για να γίνει το UI πιο user-friendly. Στη συνέχεια, προστίθεται ένα <script> στο profile view που καλεί το αρχείο profile.js μέσα από την src ιδιότητα του, καθώς και ένα ενσωματωμένο <script> στο view που κατά το register του service worker τσεκάρει το compatibility της push υπηρεσίας με τον browser. Εάν είναι συμβατά τότε θα τσεκάρει εάν ο συνδεδεμένος χρήστης έχει εγγραφεί στην υπηρεσία. Εάν ναι, θα ανανεώνεται η κατάσταση εγγραφής:

```
<script src='/profile.js'></script>
<script>
     // Service Worker Registration
    let swReg;

    // Register Service Worker
    navigator.serviceWorker.register('sw.js')
    .then( registration => {

          // Reference registration globally
          swReg = registration;

          // Check if a subscription exists, and if so, update the UI
          swReg.pushManager.getSubscription().then( setSubscribedStatus );

        // Log errors
    }).catch( error => {
        console.log(error);
        toastr.error( null, "This App is not supported by your browser.",
{timeout: 3000});
    });

</script>
```

Δημιουργείται το αρχείο profile.js μέσα στο φάκελο app, και προστίθεται μέσα στο νέο αρχείο ο κώδικας:

```
// Update UI for subscribed status
const setSubscribedStatus = (state) => {

  if (state) {
```

```
    document.getElementById('subscribe').className = 'hidden';
    document.getElementById('unsubscribe').className = '';
  } else {
    document.getElementById('subscribe').className = '';
    document.getElementById('unsubscribe').className = 'hidden';
  }
}
```

Η μέθοδος setSubscribedStatus() εναλλάσσει μεταξύ της συνδρομητικής και της μη εγγραφόμενης κατάστασης του client χρησιμοποιώντας την παράμετρο 'state' και εφαρμόζει το αποτέλεσμα στα buttons καθιστώντας τα ορατά ή κρυμμένα, καθορίζοντας επίσης και την κλάση τους.

Για την κατασκευή του push διακομιστή δημιουργείται στο φάκελο /server το αρχείο push_configuration.js και το αρχείο push.js. Η υπηρεσία push έρχεται σε σύγκρουση με την πολιτική πρόσβασης-ελέγχου του browser αφού ο push server τρέχει πάνω στον localhost, σε διαφορετική θύρα από την 8888 που τρέχει ο Express. Χρειάζεται το CORS (Cross-Origin Resource Sharing) ενεργοποιημένο για να αναγκάσει την πολιτική πρόσβασης-ελέγχου να παρακάμψει το πρόβλημα. Γι' αυτό το σκοπό στέλνεται ένα αίτημα HTTP με μια επικεφαλίδα στο response αντικείμενο που θα προκύψει από τη μέθοδο http.createServer(). Το CORS ενεργοποιείται αφού εκτελεστεί η response.setHeader('Access-Control-Allow-Origin', '*');
Μέσα στο /server/push_configuration.js λοιπόν, προστίθεται ο κώδικας που κατασκευάζει τον server και τα routes που αφορούν τα επιμέρη στοιχεία του request αντικειμένου στην υπηρεσία push που θα χρειαστούν:

```
const http = require('http');
http.createServer( (request, response) => {

    // Enable CORS
    response.setHeader('Access-Control-Allow-Origin', '*');

    // Get request vars
    const { url, method } = request;

    // 1. Subscribe
    if ( method === 'POST' && url.match(/^\/subscribe\/?/) ) {

        // Get POST Body
        let body = [];

        // Read body stream
```

```
        request.on( 'data', chunk => body.push(chunk) ).on( 'end', () => {

          // Parse subscription body to object
          let subscription = JSON.parse(body.toString());

          // Store subscription for push notifications
          push.addSubscription( subscription );

          // Respond
          response.end('Subscribed');
        })

      // 2. Public Key
      } else if ( url.match(/^\/key\/?/) ) {

        // Respond with public key from push module
        response.end( push.getKey() );

      // 3. Push Notification
      } else if ( method === 'POST' && url.match(/^\/push\/?/) ) {

        // Get POST Body
        let body = [];

        // Read body stream
        request.on( 'data', chunk => body.push(chunk) ).on( 'end', () => {
          push.send( body.toString() );
          response.end('Push sent');
        })
      // 4. Not Found
    } else {
      response.status = 404;
      response.end('Error: Unknown Request');
    }
}).listen( 3333, () => { console.log('Push Server Running on 3333') });
```

Routes:

1. Εάν η μέθοδος ισούται με POST και χρησιμοποιώντας ένα regular expression, εαν το endpoint της διεύθυνσης URL αντιστοιχεί στο '/subscribe', αφού αναγνωριστεί, θα πρέπει να ανακτηθεί το σώμα δεδομένων της αίτησης μετάδοσης ως ροή (stream). Δημιούργηθηκε λοιπόν, ένας νέος πίνακας για να κρατήσει τα δεδομένα, να διαβάσει τη ροή του σώματος παρατηρώντας παράλληλα

το αίτημα για το data event εάν πυροδοτήθηκε. Έτσι, προκείπτει μια callback που μπορεί να ωθήσει τα δεδομένα επάνω στη διάταξη του σώματος και έπειτα να τα τοποθετήσει πάνω στο αίτημα και τελικά να παρατηρήσει το end event. Το αίτημα έχει ληφθεί και η ροή του σώματος έχει ολοκληρωθεί - σε αυτό το σημείο δεν μπορεί να χρησιμοποιηθεί μόνο το σώμα του αιτήματος, αλλά ο client να λάβει ανταπόκριση με μια "Subscribed" ειδοποίηση σε μορφή κειμένου.

2. Το επόμενο endpoint είναι ένα αίτημα GET για το δημόσιο κλειδί. Η λογική της συνδρομής είναι πως για να εφαρμοστεί η push υπηρεσία, χρειάζεται τα κλειδιά που θα δημιουργηθούν και τα οποία χρησιμοποιούνται από τον browser για να εγγραφούν στο διακομιστή push. Για αυτό το αίτημα δίνεται το δημόσιο κλειδί.

3. Στη συνέχεια δημιουργείται το route για την αποστολή ειδοποίησης ώθησης σε όλους τους εγγεγραμμένους clients. Αυτό πιθανότατα δεν είναι ένα κοινό σημείο πρόσβασης στο διακομιστή, αλλά εξαρτάται πραγματικά από τη φύση της εφαρμογής. Υποδεικνύεται πώς ο διακομιστής HTTP χρησιμοποιεί ειδοποιήσεις push. Αν η μέθοδος είναι POST και καθώς το response body υποβολής του request θα κρατήσει το μήνυμα ειδοποίησης - και το endpoint του URL αντιστοιχεί στο '/push', διαβάζοντας το POST body ακριβώς με τον ίδιο τρόπο όπως στο request του 'Subscribe'. Αυτή τη φορά γίνεται ανταπόκριση με το μήνυμα "Push Sent" και καλείται η send() που πρόκειται να ολοκληρωθεί στη συνέχεια.

4. Τέλος, μια επανάκληση για οποιοδήποτε αίτημα που δεν ταιριάζει με μία από τις παραπάνω επιλογές θα αναγνωρίζεται πως δεν βρέθηκε με τον ανάλογο κωδικό κατάστασης. Ορίστηκε ο κωδικός κατάστασης σε 404 και επιστρέφεται το μήνυμα σφάλματος «Unknown Request».

Στον κώδικα του profile.js προστίθεται:

```
// Push Server url
const serverUrl = 'http://localhost:3333';

// Get public key
const getApplicationServerKey = () => {

  // Fetch from server
  return fetch(`${serverUrl}/key`)

    // Parse response body as arrayBuffer
    .then( res => res.arrayBuffer() )

    // Return arrayBuffer as new UInt8Array
```

```
    .then( key => new Uint8Array(key) )
}

// Subscribe for push notifications
const subscribe = () => {

  // Check registration is available
  if ( !swReg ) return console.error('Service Worker Registration Not Found')

  // Get applicationServerKey from push server
  getApplicationServerKey().then( applicationServerKey => {

    // Subscribe
    swReg.pushManager.subscribe( {userVisibleOnly: true, applicationServerKey} )
      .then( res => res.toJSON() )
      .then( subscription => {

        console.log(subscribe)

         // Pass subscription to server
        fetch(`${serverUrl}/subscribe`, { method: 'POST', body:
JSON.stringify(subscription) })
          .then(setSubscribedStatus)
          .catch(unsubscribe)

      // Catch subscription error
    }).catch(console.error)
  })
}

// Unsubscribe from push service
const unsubscribe = () => {

  // Unsubscribe & update UI
  swReg.pushManager.getSubscription().then( subscription => {
    subscription.unsubscribe().then( () => {
      setSubscribedStatus(false)
    })
  })
}
```

Ορίζεται η μεταβλητή 'serverUrl' να έχει τιμή 'http://localhost:3333' και

ορίζεται η μέθοδος subscribe() καθώς ελέγχεται εάν είναι διαθέσιμη η υπηρεσία push από τον service worker. Εάν δεν είναι, επιλέγεται το rejected promise με το .catch() και παρουσιάζεται σφάλμα. Διαφορετικα, εφαρμόζεται η λειτουργία του. Απαιτούνται δύο παράμετροι: userVIsibleOnly: true και applicationServerKey που είναι το δημόσιο κλειδί που ζητείται από το διακομιστή. Ορίζεται μια νέα λειτουργία getApplicationServerKey() για την ανάκτηση του κλειδιού από το διακομιστή. Εκείνη με τη σειρά της επιστρέφει ένα promise από τη μέθοδο fetch που θα κάνει request ο server στο endpoint `/key`. Μόλις επιλυθεί το fetch, επιστρέφεται η απάντηση του διακομιστή. Έπειτα, περνάει την απάντηση ως κείμενο και επιστρέφει το promise. Λαμβάνοντας την απάντηση της push υπηρεσίας, επιστρέφεται στην αλυσίδα των promises ως αντικείμενο JSON καλώντας res.toJSON(). Από τη στιγμή που θα επιλυθεί το push object θα πρέπει να σταλεί στον Push server για να λάβει μέρος η εγγραφή ώστε να εμφανίζονται οι ειδοποιήσεις push. Θα γίνει μετάδοση αυτού του αντικειμένου συνδρομής στον διακομιστή ώθησης χρησιμοποιώντας το fetch.

Τέλος, ορίζεται η unsubscribe() μέθοδος που αλλάζει το subscribed status σε false και διαγράφει τον client από την υπηρεσία. Κάθε φορά που χρειάζεται απαλλαγή από αυτή την ενεργή συνδρομή της υπηρεσίας push, πρέπει να γίνει επαναφορά των δικαιωμάτων push ειδοποιήσεων του προγράμματος περιήγησης κάνοντας κλικ στο κουμπί διαγραφής.
Η getKey() μέθοδος θα δηλωθεί στη συνέχεια στο άδειο προς το παρόν αρχείο server/push.js που αφορά το push module της εφαρμογής. Εγκαθίσταται στο project το web-push module με την εντολή `yarn add web-push`. Έπειτα εκτελείται η εντολή στον terminal:

```
node_modules/web-push/src/cli.js generate-vapid-keys --json > server/vapid.json
```

Με αυτόν τον τρόπο, δημιουργούνται τα κλειδιά και αποθηκεύονται σε ένα αρχείο server/vapid.json. Το τελευταίο βήμα στην προετοιμασία του κλειδιού είναι να μεταβιβαστεί ως Uint8Array τύπος (απαιτούμενος) στο πρόγραμμα περιήγησης. Στη μέθοδο getApplicationServerKey() που δηλώθηκε προηγουμένως ο κώδικας κάνει ακριβώς αυτό.
Καλείται το push.js στο push_configuration.js κάτω από το const http:

```
const push = require(`${__dirname}/../server/push.js`);
```

Μέσα στο push.js στη συνέχεια, προστίθεται ο κώδικας:

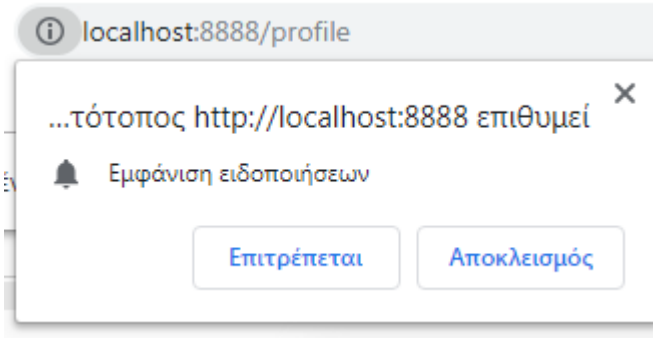
```
 // Modules
const webpush = require('web-push');
const urlsafeBase64 = require('urlsafe-base64');

// Vapid Keys
```

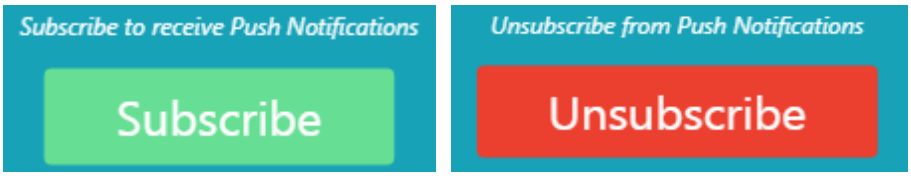
```
const vapid = require('../vapid.json');

//Configure web-push
webpush.setVapidDetails(
    'mailto:stefanoskokkalhs@gmail.com',
    vapid.publicKey,
    vapid.privateKey
);

// Create URL safe vapid public key
module.exports.getKey = () => urlsafeBase64.decode( vapid.publicKey );
```

Το κομμάτι του κλειδιού είναι έτοιμο και έτσι ο client μπορεί να γίνει συνδρομητής στον Push server. Πρώτα συνδέονται τα αρχεία push.js και push_configuration.js με το /server/index.js και γίνονται προσθήκη στον κώδικα του index.js, κάτω από το const server = http.Server(app), οι γραμμές:

```
// CREATE PUSH SERVER
const pushServer = require(`${__dirname}/../server/push_configuration.js`)
```

Τρέχοντας την εφαρμογή και μετά το register του νέου service worker, πατώντας το κουμπί subscribe ο browser ζητάει άδεια απο τον χρήστη για την εμφάνιση των ειδοποιήσεων καθώς και το κουμπί ανανεώνεται σε unsubscribe:





Επόμενο βήμα είναι η δημιουργία ενός αποθηκευτικού χώρου που θα συντηρεί τις συνδρομές. Για το σκοπό αυτό εγκαθίσταται το node_storage module.

Στη συνέχεια, δημιουργείται ένα νέο instance αποθήκευσης από αυτό το module - αυτό είναι το αρχείο στο οποίο θα εγγραφούν οι συνδρομές. Αυτή η ενότητα αποθηκεύει αντικείμενα JavaScript ή δεδομένα JSON.

Αυτό που πρέπει να γίνει είναι όταν ξεκινάει ο διακομιστής, να ελέγξει πρώτα τις υπάρχουσες συνδρομές στο storage και να τεθεί η συστοιχία συνδρομών στον storage πίνακα. Με την store.get('subscriptions') επιστρέφεται το value του κλειδιού του πίνακα των συνδρομών αν εμπεριέχει στοιχεία ή null αν ο πίνακας είναι άδειος. Για να κατασταθεί η διαδικασία συνεχής, προστίθεται η μέθοδος addSubscription(). Η εγγραφή αποθηκεύεται στον πίνακα και στη συνέχεια ενημερώνεται το storage με τη νέα συστοιχία, χάρις την store.put ('subscriptions', subscriptions). Ανανεώνεται το push.js προσθέτοντας τη λειτουργία addSubscription().

```
// Modules

const webpush = require('web-push');

const urlsafeBase64 = require('urlsafe-base64');

const Storage = require('node-storage');


// Vapid Keys

const vapid = require('../vapid.json');


//Configure web-push

webpush.setVapidDetails(

    'mailto:stefanoskokkalhs@gmail.com',

    vapid.publicKey,

    vapid.privateKey

);


// Subscriptions

const store = new Storage(`${__dirname}/nodeDB`);

let subscriptions = store.get('subscriptions') || [];


console.log(subscriptions);
```

```
// Create URL safe vapid public key
module.exports.getKey = () => urlsafeBase64.decode( vapid.publicKey );

// Store new subscription
module.exports.addSubscription = (subscription) => {

  // Add to subscriptions array
  subscriptions.push(subscription);

  // Persist subscriptions
  store.put('subscriptions', subscriptions);
}
```

Τέλος, πρέπει να κατασκευαστεί η μέθοδος send() μέσα από το module του push.js. Προστίθεται στο τέλος του αρχείου:

```
//Send notification to all registered subscriptions
module.exports.send = (message) => {
    //Notification promises
    let notifications = [];

    subscriptions.forEach( (subscription, i) => {
        let p = webpush.sendNotification( subscription, message ).catch( status
=> {

            // Check for "410 - Gone" status and mark for deletion
            if (status.statusCode === 410) subscriptions[i]['delete'] = true;

            //Return any value
            return null;
        });
```

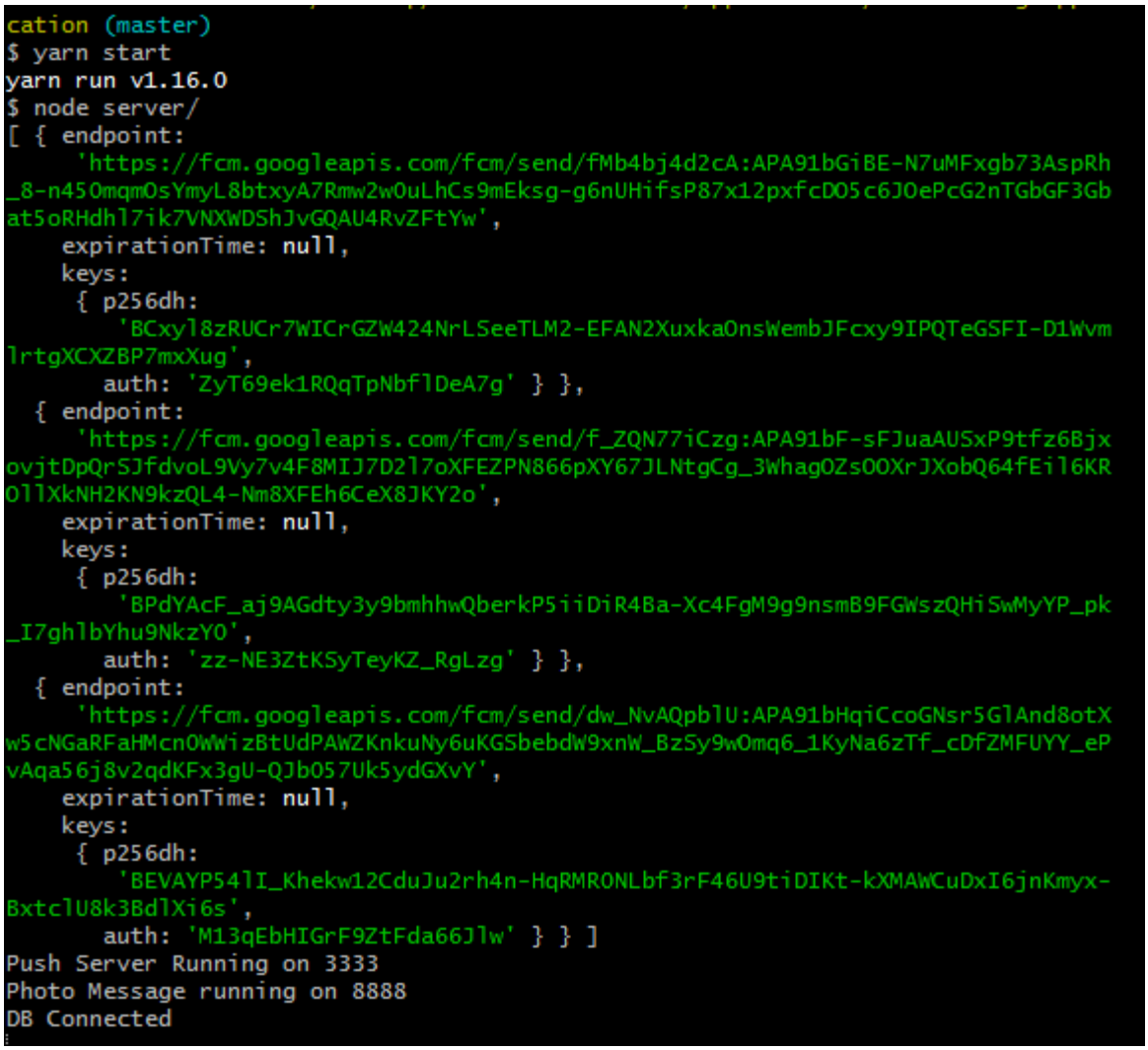
```
        //Push notification promise to array
        notifications.push(p);
    })
    //Clean subscriptions marked for deletion
    Promise.all(notifications).then( () => {
        //filtering
        subscriptions = subscriptions.filter( subscription =>
!subscription.delete );

        //Persist 'cleaned' subscriptions
        store.put( 'subscriptions', subscriptions);
    })
}
```

Με την send(message) στέλνεται για κάθε subscription ένα notification κάθε φορά που γίνεται push από τον server. Δηλώνεται ένας άδειος πίνακας notifications = []. Έπειτα δημιουργείται μία λούπα με την forEach() κάνοντας προσπέλαση τον πίνακα subscriptions. Για κάθε συνδρομή, συνδέεται η webpush.sendNotification() με τη δήλωση της μεταβλητής p, που πρόκειται να αποθηκεύει το αποτέλεσμα του επιστρεφόμενου promise της κάθε sendNotication() που θα εκτελείται στη λούπα.

Στην περίπτωση όμως που ένας χρήστης, ενώ ήταν αρχικά συνδρομητής πάτησε μετά το unsubscribe κουμπί και έπαψε να είναι εγγεγραμμένος, όταν λάβει notification τότε το promise γίνεται rejected. Με την .catch() παρέχεται πρόσβαση στο σφάλμα και επιστρέφεται το status code το οποίο, σε αυτή την περίπτωση δεν είναι 404 αλλά 501 και αφορά την κατάσταση `Gone`. Δίνεται στο `delete` property του συγκεκριμένου subscription η τιμή true και έτσι εκείνο “μαρκάρεται” για να χειραγωγηθεί στη συνέχεια. Τέλος επιστρέφεται null. Έξω από την .catch(), προστίθεται σε κάθε λούπα η τιμή του p, η επιστροφή δηλαδή του εκάστοτε promise.

Στο τέλος, με την Promise.all(notifications) επιστρέφεται ένα promise όταν ολοκληρωθούν όλα τα υπόλοιπα promises που γίνονται προηγουμένως στη λούπα της forEach(). Αφού λοιπόν ο πίνακας notifications είναι γεμάτος, με τη μέθοδο .filter() φιλτράρονται τα subscriptions και επιστρέφονται εκείνα που δεν είναι μαρκαρισμένα με την “‘delete’: true” ιδιότητα καθώς αποθηκεύεται και η τελική σύζευξη τους στο storage.

Εκτελώντας την εντολή yarn start στον terminal ο express server ξεκινάει τη λειτουργία του και βρίσκεται στην πόρτα 8888, ταυτόχρονα ξεκινάει και ο push server στην 3333. Παρατηρούνται όλα τα subscriptions στον terminal.



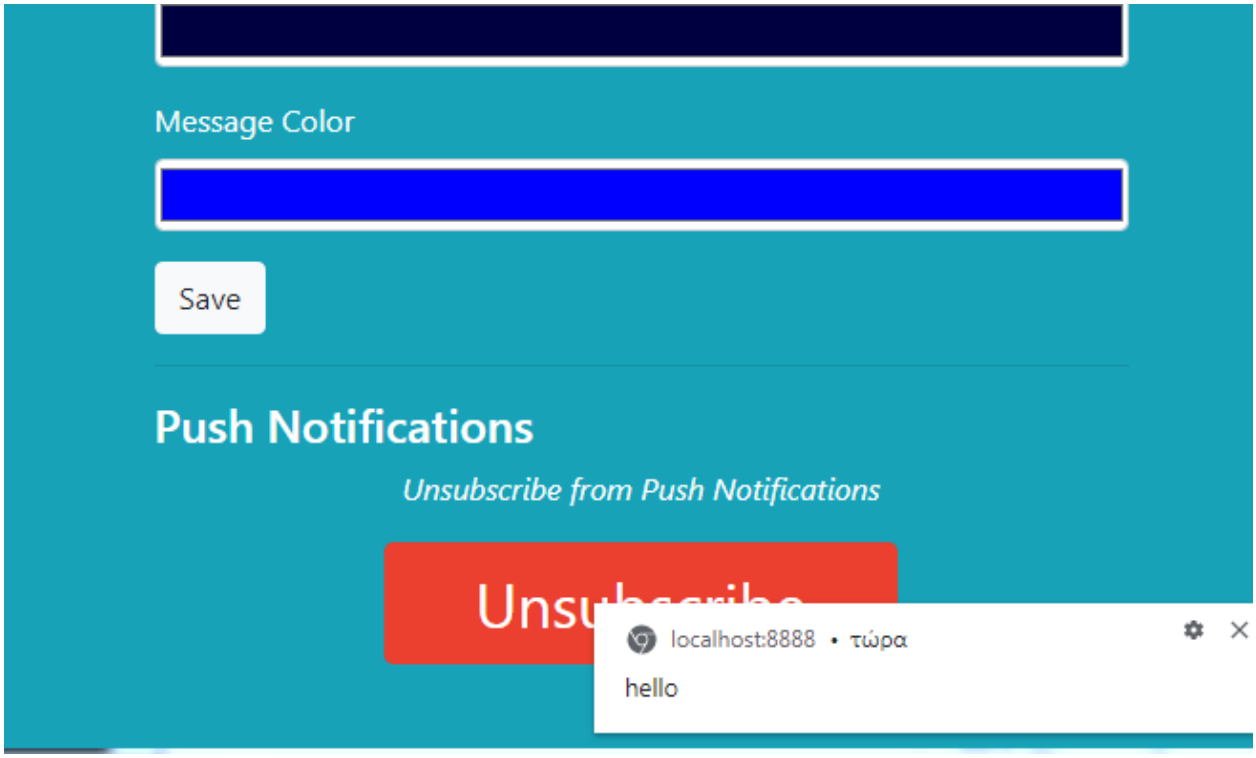
Ανοίγοντας έναν δεύτερο terminal, πηγαίνοντας στο φάκελο της εφαρμογής τρέχει η εντολή `curl POST -d 'hello' localhost:3333/push`.

Με την curl μέσα από το τερματικό πραγματοποιούνται εύκολες API κλήσεις. Μπορούν να χρησιμοποιηθούν διάφορα web tools για έναν τέτοιο σκοπό με πιο γνωστό να είναι το POSTMAN. Στέλνεται μία ειδοποίηση push με το μήνυμα ΄hello΄ που ουσιαστικά είναι ένα POST request στον server. Ο push server θα συμπεριφερθεί όπως ακριβώς έχει προγραμματιστεί η συμπεριφορά του στο push_configuration.js, και συγκεκριμένα στο σημείο “if (method === 'POST' && url.match(/^\/push\/?/))”.

Θα σταλεί σε όλους τους συνδρομητές το notification με body ‘hello’.

```
  0     0    0     0    0     0      0      0 --:--:--  0:00:02 --:--:--     0cu
rl: (6) Could not resolve host: X-POST
curl: (7) Failed to connect to localhost port 3333: Connection refused

User@User-PC MINGW64 ~/Desktop/ΠΤΥΧΙΑΚΗ 2018-2019/Applications/PhotoMessageAppli
cation (master)
$ curl X-POST -d 'hello' localhost:3333/push
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:--  0:00:02 --:--:--     0curl: (6) Coul
d not resolve host: X-POST
100    14  100     9  100     5    204    113 --:--:-- --:--:-- --:--:--   317Push sent

User@User-PC MINGW64 ~/Desktop/ΠΤΥΧΙΑΚΗ 2018-2019/Applications/PhotoMessageApplication (mast
er)
$
```

Push sent



Στην περίπτωση που ο client δεν είναι συνδρομητής ή δεν έχει δώσει πρόσβαση στα push notifications του browser, τότε το Notification δεν θα εμφανιστεί.

9.11 App Manifest Και Testing

Η τελευταία προσθήκη στο app για να μετατραπεί ολοκληρωμένα σε ένα Progressive Web App είναι το app manifest. Μέσα στο φάκελο /app δημιουργείται το αρχείο manifest.json και μέσα προστίθεται:

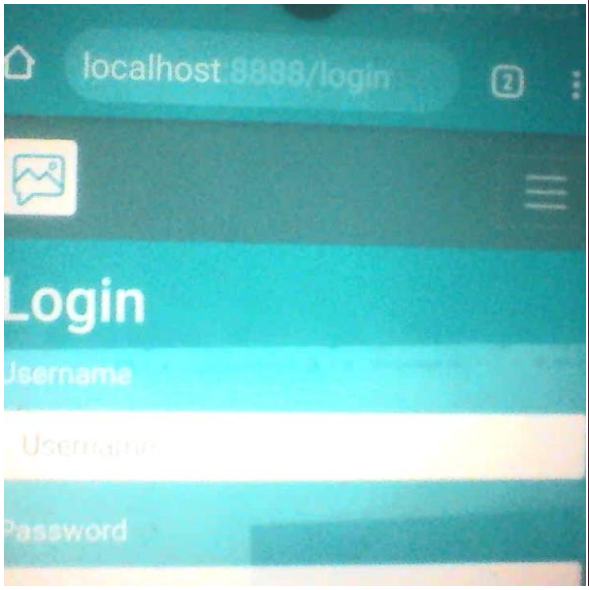
```
{
  "name": "PhotoChatroom",
  "short_name": "PhotoMsgChatRoom",
  "theme_color": "#19a2b8",
  "background_color": "#19a2b8",
  "display": "standalone",
  "orientation": "portrait",
  "Scope": "/",
  "start_url": "/",
  "icons": [
    {
      "src": "/icons/icon-72x72.png",
      "sizes": "72x72",
      "type": "image/png"
    },
    {
      "src": "/icons/icon-96x96.png",
      "sizes": "96x96",
      "type": "image/png"
    },
    {
      "src": "/icons/icon-128x128.png",
      "sizes": "128x128",
      "type": "image/png"
    },
    {
```

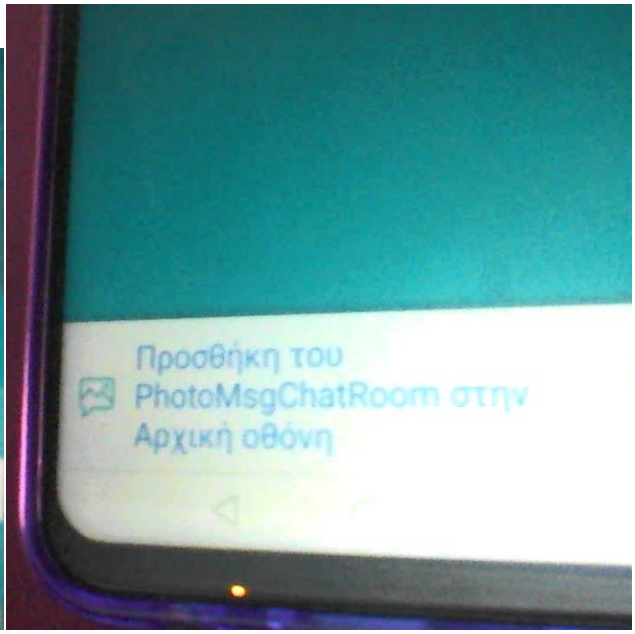
```
      "src": "/icons/icon-144x144.png",
      "sizes": "144x144",
      "type": "image/png"
    },
    {
      "src": "/icons/icon-152x152.png",
      "sizes": "152x152",
      "type": "image/png"
    },
    {
      "src": "/icons/icon-192x192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "/icons/icon-384x384.png",
      "sizes": "384x384",
      "type": "image/png"
    },
    {
      "src": "/icons/icon-512x512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ],
  "splash_pages": null
```

}

Το αρχείο έγινε generate μέσα από το online manifest generator μαζί με τα icons που προστέθηκαν στο φάκελο /app. Το splash_pages μένει κενό (null) και αυτόματα θα γίνεται generate ένα splash screen κατά την εκκίνηση του app στη mobile συσκευή που βρίσκεται εγκατεστημένο. Το splash screen λειτουργεί ως μια εικόνα καλωσορίσματος και είναι συνηθισμένο χαρακτηριστικό των native εφαρμογών εώς να ολοκληρωθεί η φόρτωση τους. Έχει στο κέντρο της το icon της εφαρμογής, τον τίτλο και από πίσω έχει το χρώμα του background που είναι δηλωμένο στο app manifest αρχείο. Επίσης, είναι χρονικά προγραμματισμένη να γίνεται hide μετά από 1 δευτερόλεπτο.

Τρέχοντας την εφαρμογή από τα remote devices του chrome και το USB debugging και στη συνέχεια ανοίγοντας από το smartphone τον chrome και δίνοντας localhost:8888 στο URL ενώ παράλληλα τρέχει ο server από τον terminal στο desktop, θα εμφανιστεί ο ιστότοπος. Με την εισαγωγή του χρήστη στη σελίδα γίνεται redirect στο /login endpoint και view. Επίσης, εμφανίζεται στο κάτω μέρος της οθόνης ένα αυτοδημιούργητο παράθυρο ρωτώντας το χρήστη εάν επιτρέπει την εγκατάσταση της εφαρμογής στο κινητό.



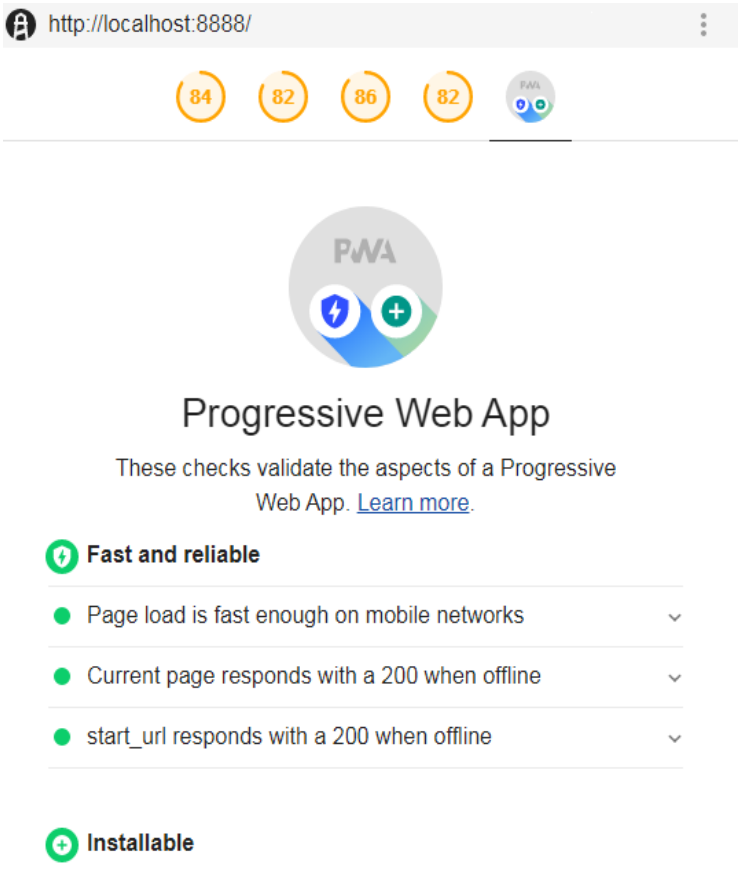


Εγκαθιστώντας την εφαρμογή παρατηρείται το εικονίδιο στο homescreen της συσκευής. Τρέχοντας την, γίνεται προβολή του splash screen και η εφαρμογή ανοίγει στο login view χωρίς τη γραμμή εισαγωγής URL από πάνω, υποδηλώνοντας έτσι πως ο ιστότοπος έχει φύγει από τον browser και είναι πλέον μία ανεξάρτητη εφαρμογή.

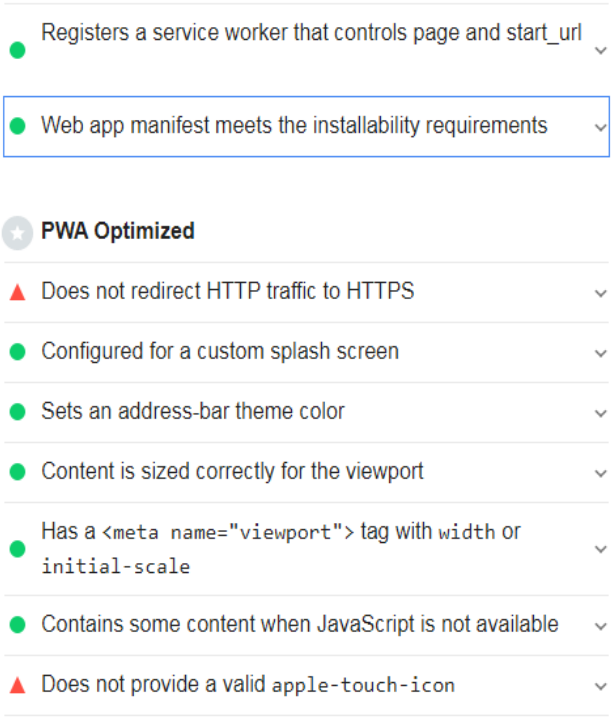
Στην εφαρμογή τώρα, υπάρχει η δυνατότητα να γίνει register νέος χρήστης, να κάνει login ένας πιστοποιημένος χρήστης, να αλλάξει τα χρώματα του chat στο /profile και να έχει πρόσβαση στο chat room και στη κάμερα του κινητού, καθώς υπάρχει ο service worker που αναλαμβάνει την αποθήκευση του app shell στην κρυφή μνήμη και την φόρτωση πόρων κατά την offline εργασία.

Επίσης υπάρχει το authentication feature για την εξουσιοδότηση πρόσβασης στους χρήστες κατά το session τους. Όλα λειτουργούν κανονικά εκτός από το subscribe() κατά το tap στο subscribe button μέσα στο profile view. Αυτό συμβαίνει γιατί επιστρέφεται ένα fetch από το localhost:3333/key που κάνει conflict με το localhost του Android προσομοιωτή. Χρειάζεται να γίνει εισαγωγή στατικής IP στη θέση του localhost κατα τη δήλωση της μεταβλητής `serverUrl` και οι ειδοποιήσεις λειτουργούν κανονικά(π.χ. 88.88.1.107 αντί του «localhost»).

Μέσα από το lighthouse μπορεί κανείς να εξετάσει εφαρμογές ως προς το performance μέσα από έναν test προσομοιωτή που αναζητεί προβλήματα και αναλύει τον κώδικα με βάση πρότυπα χρήσης. Εγκαθίσταται στον chrome σαν chrome add-on και πατώντας το εικονίδιο πάνω δεξιά >  ενώ η τοποθεσία είναι στη διεύθυνση localhost:8888, εκτελείται το τεστ. Σε αυτό το στάδιο δεν έχει πιστοποιηθεί η εφαρμογή να λειτουργεί στο HTTPS πρωτόκολλο οπότε δεν πρόκειται το τελικό score να είναι άριστο.



Παρ' όλα αυτά, κατασκεύαστηκε μια εφαρμογή που στοχεύει στην απόδοση, λαμβάνοντας υπ όψιν τις σημαντικές απαιτήσεις στο κομμάτι του UX καθώς και παρουσιάστηκε η απλή ανάπτυξη της βήμα - βήμα χρησιμοποιώντας μοντέρνες τεχνολογίες που επιτρέπουν την πρόσβαση στις native δυνατότητες των mobile εφαρμογών. Η εφαρμογή καταλήγει να ανήκει άξια στην κατηγορία των PWA. Ενώ κατασκευάστηκε σαν web εφαρμογή μπορεί να λειτουργήσει offline και σε mobile. Επίσης, έχει την ικανότητα να εγκατασταθεί και να είναι προσβάσιμη από το homescreen της mobile συσκευής καθώς και να λαμβάνει push notifications από τον push server.



Μπορεί το project να διαμοιραστεί σε developers, κάνοντας clone από το Git repository που παρουσιάζεται παρακάτω.

Τα μόνα requirements είναι:

1. Η αλλαγή των στοιχείων που αφορούν την είσοδο στη βάση στο db/db.js και τα στοιχεία του options για το storage μέσα στο server/index.js, που αφορούν τον κάθε developer.
2. Η εγκατάσταση όλων των node-modules εκτελώντας την εντολή yarn add.

https://github.com/StefanosKok/progressiveWebApp

Επίσης, το project έχει πιστοποιηθεί σε https αλλά δεν είναι συνεχώς online. Η επίσκεψη σε αυτό γίνεται μέσα από τη διεύθυνση:

https://wasteland.ddns.net:444

Βιβλιογραφία

[*α*], σελ 19: Comscore's "The 2017 US Mobile App Report" https://www.comscore.com/Insights/Presentations-and-Whitepapers/2017/The-2017-US-Mobile-App-Report,

[*β*], σελ 29: "Most Us Smartphone Users Download Zero Apps per Month" https://www.leanplum.com/blog/app-usage/,

[*γ*], σελ 76: "USB Remote Debugging How-to 1"https://developers.google.com/web/tools/chrome-devtools/remote-debugging,

[*δ*], σελ 76: "USB Remote Debugging How-to 2"https://developers.google.com/web/tools/chrome-devtools/remote-debugginghttps://stackoverflow.com/questions/21925992/chrome-devtools-devices-does-not-detect-device-when-plugged-in,

https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/,

https://en.wikipedia.org/wiki/Progressive_web_applications,

https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS,

https://nodejs.org/api/http.html,

https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages,

https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html,

https://socket.io/,

https://scotch.io/tutorials/learn-to-use-the-new-router-in-expressjs-4,

https://github.com/expressjs/express,

https://blog.codinghorror.com/understanding-model-view-controller/,

https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller,

https://developers.google.com/web/progressive-web-apps/,

https://app-manifest.firebaseapp.com/,

https://love2dev.com/blog/service-worker-cache/,

https://developers.google.com/web/fundamentals/instant-and-offline/offline-cookbook/,

https://serviceworke.rs/caching-strategies.html,

https://dev.to/paco_ita/service-workers-and-caching-strategies-explained-step-3-m4f,

https://developers.google.com/web/tools/chrome-devtools/remote-debugging/,

https://developers.google.com/web/fundamentals/web-app-manifest/,

https://medium.com/devc-kano/basics-of-authentication-using-passport-and-jwt-with-sequelize-and-mysql-database-748e09d01bab,

https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia#Browser_compatibility,

http://www.passportjs.org/docs/authenticate/,

https://stackoverflow.com/questions/33897276/what-is-the-difference-between-a-session-store-and-database,

https://www.npmjs.com/package/express-session,

https://www.npmjs.com/package/express-mysql-session,

https://github.com/srinathgs/mysqlstore,

https://codelabs.developers.google.com/codelabs/add-to-home-screen/#0,

https://www.freecodecamp.org/news/how-to-get-https-working-on-your-local-development-environment-in-5-minutes-7af615770eec/,

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise,

https://en.wikipedia.org/wiki/Ajax_(programming),

https://github.com/nodejs/node,

https://www.spiderwriting.co.uk/static-dynamic.php,

https://www.pluralsight.com/blog/film-games/whats-difference-front-end-back-end,

https://en.wikipedia.org/wiki/Front-end_web_development,

https://www.techopedia.com/definition/29568/back-end-developer,

https://searchnetworking.techtarget.com/definition/TCP-IP,

https://en.wikipedia.org/wiki/Internet,

https://getbootstrap.com/docs/4.2/getting-started/introduction/,

https://api.jquery.com/,

https://www.seguetech.com/client-server-side-code/,

https://ionicframework.com/resources/whitepapers/pwa-architects-guide,

https://hackernoon.com/everything-you-need-to-know-about-progressive-web-app-pwa-6524edbb0c57,

https://www.altexsoft.com/blog/engineering/progressive-web-apps/,

https://www.freecodecamp.org/news/write-less-do-more-with-javascript-es6-5fd4a8e50ee2/,

https://www.w3schools.com/js/js_es6.asp,

https://handlebarsjs.com/,

https://www.atlassian.com/git/tutorials/git-bash,

https://developers.google.com/web/tools/chrome-devtools/progressive-web-apps

https://developer.mozilla.org/en-US/docs/Web/API/Push_API,

https://www.urbanairship.com/push-notifications-explained?utm_source=googleplus_sumo_share&utm_medium=website&utm_campaign=ua_web,

https://vwo.com/push-notifications/,

https://w3.org/TR/messaging/,

https://developer.mozilla.org/en-US/docs/Web/API/notification,

https://firebase.google.com/,

https://developers.google.com/web/tools/workbox/,

https://developers.google.com/web/fundamentals/architecture/app-shell,

https://css-tricks.com/how-to-get-a-progressive-web-app-into-the-google-play-store/,

https://developers.google.com/web/fundamentals/app-install-banners/,

https://medium.com/appmaker-xyz/how-to-convert-pwa-to-an-android-app-f06bac5d2e29,

https://www.androidauthority.com/progressive-web-apps-1005564/,

https://gist.github.com/subfuzion/08c5d85437d5d4f00e58,

https://www.getpostman.com/,

https://developers.google.com/web/tools/lighthouse/

Πίνακας Συντομογραφιών

ARPANET:	The Advanced Research Projects Agency Network
WWW:	World Wide Web
Dev:	Development / Developer
DevTools:	Developer Tools [provided by a BROWSER]
IP:	Internet Protocol
TCP:	Transmission Control Protocol
HTTP:	HyperText Transfer Protocol
HTTPS:	HyperText Transfer Protocol Secure
TLS:	Transport Layer Security
SSL:	Secure Sockets Layer
ISP:	Internet Service Provider
URL:	Uniform Resource Locator
W3C:	World Wide Web Consortium
UTF-8:	Unicode Tranformation Format 8 [8-bit blocks for representing characters]
HTML:	HyperText Markup Language
HTML5:	HyperText Markup Language 5 [NEWER VERSION]
CSS:	Cascading Style Sheet
CSS3:	Cascading Style Sheet 3 [NEWER VERSION]
JS:	JavaScript
UX:	User Experience
UI:	User Interface

PHP:	Hypertext Preprocessor [earlier Personal Home Pages]
SQL:	Stractured Query Language
DB:	Data Base
XML:	eXtensible Markup Language
AJAX:	Asynchronous JavaScript And XML
XHR:	XML Http Request
App:	Application
API:	Application Program Interface
SPA:	Single Page Application
MVC:	Model-View-Controller
PWA:	Progressive Web Application
SW:	Service Worker
ECMAScript:	European Computer Manufacturers Association Script
JSON:	JavaScript Object Notation
NPM:	NodeJS Package Manager
REQ:	Request
RES:	Response
Async:	Asynchronous
USB:	Universal Serial Bus
CORS:	Cross-Origin Resource Sharing
RegExp:	Regular Expression
ID:	Identitier
UUID:	Universal Unique Identifier

Const:	Constant
Var:	Variable
Msg:	Message